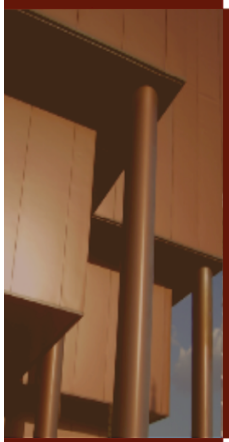
Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

Implementación de WPS en el firmware NodeMCU para el ESP8266

Autor: Julio Candelario Elías
Tutor: Antonio Luque Estepa



Dep. Ingenería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2016

Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

Implementación de WPS en el firmware NodeMCU para el ESP8266

Autor:
Julio Candelario Elías

Tutor:
Antonio Luque Estepa
Profesor titular

Dep. Ingeniería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2016

Trabajo Fin de Grado: Implementación de WPS en el firmware NodeMCU para el ESP8266

Autor: Julio Candelario Elías

Tutor: Antonio Luque Estepa

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2016

El Secretario del Tribunal

A mi familia

A mis maestros

Agradecimientos

Quisiera agradecer en primer lugar este trabajo a mis padres que me han dado la posibilidad de estudiar esta carrera y siempre me han apoyado al igual que mi hermana, han estado siempre ahí para animarme en los momentos difíciles y celebrar los buenos.

También quisiera agradecer a mi tutor Antonio Luque Estepa, por ayudarme y atenderme siempre que lo he necesitado.

Por último agradecérselo a mis amigos tanto de toda la vida como de aquí de Sevilla, a mis compañeros de clase que me han acompañado todos estos años y compañeros de piso, una mención especial al "tridente electrónico", que son con los que más momentos, trabajos, presentaciones,... he compartido, todos ellos han hecho que estos años hayan sido mucho más llevaderos.

Julio Candelario Elías

Sevilla, 2016

Resumen

Este trabajo se centra en el system on chip ESP8266, más en concreto en añadir el estándar WPS al firmware de NodeMCU.

Para llevar a cabo este trabajo se han estudiado las características de este SOC y su funcionamiento, así como el estándar WPS y sus distintos modos de funcionamiento. Se ha creado un módulo nuevo en el NodeMCU que permite el uso de las funciones de la librería “libwps.a” proporcionada por la compañía Espressif en su sdk, las cuáles permitirán al NodeMCU hacer uso de este estándar.

Por último para demostrar su funcionamiento se han hecho distintas pruebas, conectando un ESP8266 (con el NodeMCU modificado) a un router mediante el estándar WPS.

Índice

Agradecimientos **ix**

Resumen **xi**

Índice **xiii**

Índice de Tablas **xvi**

Índice de Figuras **xviii**

Notación **xx**

1 Introducción y objetivos **1**
1.1 Motivación *1*
1.2 Objetivos *1*
1.3 Estructura del trabajo *1*

2 Estado del arte **3**
2.1 SOC en el Mercado *3*
2.1.1 ESP8266 3
2.1.2 EMW3165 5
2.2 Programando el módulo ESP8266 *6*
2.2.1 Comandos AT 6
2.2.2 Programando el ESP8266 como un Arduino 7
2.2.3 NodeMCU 8
2.3 WPS *10*

3 Entorno de desarrollo **11**
3.1 Eclipse *11*
3.1.1 Configuración de eclipse 12
3.2 Primeras pruebas (usando Eclipse) *15*
3.2.1 Modo flash del ESP8266 16
3.2.2 Hello World 16
3.2.3 Blinky LED 17
3.3 Docker *18*
3.3.1 Compilar el firmware NodeMCU 19
3.3.2 Flashear el firmware de NodeMCU 21
3.4 Ejemplos con el NodeMCU *22*
3.4.1 Apagar y encender un LED 22
3.4.2 Conectarse a una red WiFi 23
3.4.3 Crear un servidor web para apagar y encender un LED 23

4 Realización y desarrollo **25**
4.1 WPS para el ESP8266 *25*
4.2 Como crear el nuevo módulo WPS *33*
4.2.1 Integrar el módulo en NodeMCU 34
4.3 Construyendo el módulo WPS *36*

5 Pruebas **41**
5.1 Entorno de pruebas *41*
5.2 Pruebas *42*
5.2.1 Escenario 1: Situación normal 42
5.2.2 Escenario 2: Se apaga el ESP8266 durante la conexión 45
5.2.3 Escenario 3: Se apaga el router durante la conexión 45
6 Conclusiones y desarrollos futuros **47**
6.1 Conclusiones *47*
6.2 Desarrollos futuros *48*
Referencias **50**
Anexo A: Código **52**
Anexo B: Datasheet Overview ESP8266 **60**

ÍNDICE DE TABLAS

Tabla 1 - Librerías "esp_iot_sdk_v1.5.1" 26

ÍNDICE DE FIGURAS

Figura 1 - Características ESP82266 4
Figura 2 - ESP-01 5
Figura 3 - ESP-12E 5
Figura 4 - EMW3165 6
Figura 5 - Arduino ejemplo "blink" 8
Figura 6 - Modo Flash ESP8266 16
Figura 7 - Instalación "Docker Toolbox" para Mac 18
Figura 8 - Corriendo "Docker" 19
Figura 9 - Compilando NodeMCU 20
Figura 10 - Compilación terminada 20
Figura 11 - NODEMCU Flasher 1/4 21
Figura 12 - NODEMCU Flasher 2/4 21
Figura 13 - NODEMCU Flasher 3/4 22
Figura 14 - NODEMCU Flasher 4/4 22
Figura 15 - Ejemplo WPS en eclipse 32
Figura 16 - Probando el módulo "wps" 35
Figura 17 - Añadir "libwps.a" 36
Figura 18 - wifi_wps_disable 36
Figura 19 - wifi_wps_enable 37
Figura 20 - wifi_wps_start 37
Figura 21 - wifi_set_wps_cb 38
Figura 22 - TP LINK TL-WR841ND 41
Figura 23 - Activando WPS en el router 42
Figura 24 - Añadir nuevo dispositivo 42
Figura 25 - Conectando 43
Figura 26 - Conectando el ESP8266 al router 43
Figura 27 - Conectado con éxito 44
Figura 28 - ESP8266 conectado al router 44
Figura 29 - Error en la conexión 45
Figura 30 - Datasheet Overview 1/4 60
Figura 31 - Datasheet Overview 2/4 61
Figura 32 - Datasheet Overview 3/4 62
Figura 33 - Datasheet Overview 4/4 63

Notación

IOT	Internet of Things
SOC	System on Chip
WPS	Wi-Fi Protected Setup
MCU	Microcontroller Unit
RAM	Random Access Memory
ROM	Read Only Memory
E/S	Entrada/Salida
RSIC	Reduced Instruction Set Computing
CPU	Central Processing Unit
RTOS	Real Time Operation System
MIPS	Microprocessor without Interlocked Pipeline System
GPIO	General Purpose Inputa Output
PCB	Printed Circuit Board
ARM	Advanced RISC Machine
DSP	Digital Signal Processing
USART	Universal Synchronous/Asynchronous Receiver/Transmitter
I2C	Inter Integrated Circuit
SPI	Serial Peripheral Interface
ADC	Analog to Digital Conversion
DAC	Digital to Analogue Converter
PWM	Pulse Width Modulation
SDK	Software Development Kit
LED	Light Emitting Diode
ITU-T	International Telecommunication Union
IP	Internet Protocol
ID	Identity
TCP	Transmision Control Protocol
COM	Communication Port
IDE	Integrated Development Environment
TTL	Transisto Transisto Logic
IBM	International Business Machines
API	Aplication Programming Interface
MIT	Massachusetts Instute of Technology
WLAN	Wireless Local Area Network
WPA	Wi-Fi Protected Access
SSID	Server Set Identifier

PSK	Phase Shift Keying
AP	Access Point
PIN	Personal Identification Number
PBC	Push Button Configuration
NFC	Near Field Communications
USB	Universal Serial Bus
RFID	Radio Fecuency Identification
QSS	Quick Security Setup
MinGW	Minimalist GNU for Windows

1 INTRODUCCIÓN Y OBJETIVOS

1.1 Motivación

El trabajo se centra en el system on chip ESP8266, de bajo coste, usado típicamente como módem para microcontroladores. Este SOC puede programarse directamente mediante el SDK proporcionado por la compañía Espressif, pero existen otros software para facilitar esta tarea. Entre ellos se encuentra el firmware de NodeMCU, un software que permite desarrollar aplicaciones para este SOC en un lenguaje de alto nivel como es Lua.

El sdk del dispositivo cuenta con numerosas librerías, que permiten desarrollar multitud de aplicaciones. Pero no todas las posibilidades que ofrece el sdk están incluidas en el firmware del NodeMCU. Este software se encuentra en una constante evolución, una de las carencias de NodeMCU es que no soporta el estándar WPS.

1.2 Objetivos

El objetivo de este trabajo es añadir el estándar WPS al firmware de NodeMCU, con la idea de proporcionar más herramientas para el desarrollador final de aplicaciones. Este firmware es el más habitual a la hora de desarrollar aplicaciones para el ESP8266, uno de los dispositivos más usados últimamente en IoT debido a su bajo coste, reducido tamaño, fácil disponibilidad y su gran ventaja que cuenta con Wi-Fi integrado.

Primero se estudiará como funciona el estándar WPS que viene incluido en el SDK, se probará con un ejemplo. Posteriormente se modificará el código del NodeMCU para que pueda hacer uso de esta funcionalidad WPS.

1.3 Estructura del trabajo

El trabajo constará de 5 partes, que se describen a continuación:

- Se hará un estudio del ESP8266 y de su gran competidor el EMW3165, sus características hardware, software, precio,… También se explicarán algunas de las formas de programación de estos SOC.
- A continuación se expondrá el entorno de desarrollo elegido, que permitirá desarrollar aplicaciones para programar el ESP8266 y se explicarán algunos de lo ejemplos realizados con este dispositivo.
- Posteriormente se explicará y se desarrollará el trabajo llevado a cabo para la creación y construcción de este módulo WPS y su inserción dentro del firmware del NodeMCU.
- Después se creará un entorno de pruebas sobre el que probar el dispositivo y ver el funcionamiento ante distintas situaciones.
- Finalmente se expondrán las conclusiones sobre el trabajo realizado y las posibles líneas para desarrollos futuros.

2 ESTADO DEL ARTE

2.1 SOC en el Mercado

En el mercado Existen multitud de SOC, dispositivos que intentan integrar gran parte de los módulos de un computador o cualquier otro sistema informático o electrónico en un único circuito integrado o chip.

Esta búsqueda se centra en aquellos SOC usados principalmente para IOT. Estos SOC son dispositivos de bajo coste, bajo consumo de potencia, reducido tamaño,… El más usado últimamente es el ESP8266 y su gran competidor que apareció en el mercador algunos meses después el EMW3165. A continuación se expondrán las características de cada uno ellos.

2.1.1 ESP8266

Es uno de los chip con WI-Fi integrado más usados. De dimensiones muy reducidas que varían dependiendo del modelo que se escoja, con un coste de apenas 4 o 5 dólares.

Integra el procesador Tensilica L106 de 32 bits, que presenta un consumo de energía muy bajo y un conjunto de instrucciones reducido de 16 bits (RSIC). Lo que le permite alcanzar una velocidad máxima de reloj de 160MHz. [1]

Este chip puede ser usado como una interfaz con sensores externos y otros dispositivos a través de los GPIO. Cuenta con un kit de desarrollo de software (SDK) que proporciona códigos de ejemplo para varias aplicaciones.

Dispone de todo el software necesario para la conexión 802.11, es decir, para la conexión Wi-Fi. A continuación se muestra una tabla en donde se detallan estas características:

Categories	Items	Parameters
Wi-Fi	Standards	FCC/CE/TELEC/SRRC
	Protocols	802.11 b/g/n/e/i
	Frequency Range	2.4 G ~ 2.5 G (2400M ~ 2483.5M)
	Tx Power	802.11 b: +20 dBm
		802.11 g: +17 dBm
		802.11 n: +14 dBm
	Rx Sensitivity	802.11 b: -91 dbm (11 Mbps)
		802.11 g: -75 dbm (54 Mbps)
		802.11 n: -72 dbm (MCS7)
	Antenna	PCB Trace, External, IPEX Connector, Ceramic Chip
Hardware	CPU	Tensilica L106 32-bit micro controller
	Peripheral Interface	UART/SDIO/SPI/I2C/I2S/IR Remote Control
		GPIO/ADC/PWM
	Operating Voltage	3.0 V ~ 3.6 V
	Operating Current	Average value: 80 mA
	Operating Temperature Range	-40°C ~ 125°C
	Storage Temperature Range	-40°C ~ 125°C
	Package Size	QFN32-pin (5 mm x 5 mm)
	External Interface	-
Software	Wi-Fi Mode	station/softAP/SoftAP+station
	Security	WPA/WPA2
	Encryption	WEP/TKIP/AES
	Firmware Upgrade	UART Download / OTA (via network)
	Software Development	Supports Cloud Server Development / Firmware and SDK for fast on-chip programming
	Network Protocols	IPv4, TCP/UDP/HTTP/FTP
	User Configuration	AT Instruction Set, Cloud Server, Android/iOS App

Figura 1 - Características ESP82266

Ha sido diseñado para dispositivos móviles, aparatos electrónicos portátiles y para las aplicaciones de IoT. La arquitectura de ahorro de energía cuenta con tres modos de funcionamiento (modo activo, modo de reposo y modo de sueño profundo) lo que permite diseños que funcionan con baterías durante mucho más tiempo.

Funciona en el rango de temperaturas de -40 ° C a + 125 ° C, capaz de funcionar en entornos industriales. Con las características de los chip altamente integrados y una mínima cantidad de componentes externos, el chip ofrece fiabilidad, robustez y compacidad. [2]

Existen distintas versiones del ESP8266 cada una con ciertas modificaciones que serán más adecuadas o no

dependiendo de la aplicación del usuario, se va a nombrar la más común y la que se usará en este trabajo:

- ESP-01: Este modelo suele ser el más habitual o el más extendido, incluye tres pines digitales GPIO0, GPIO2 y GPIO16.

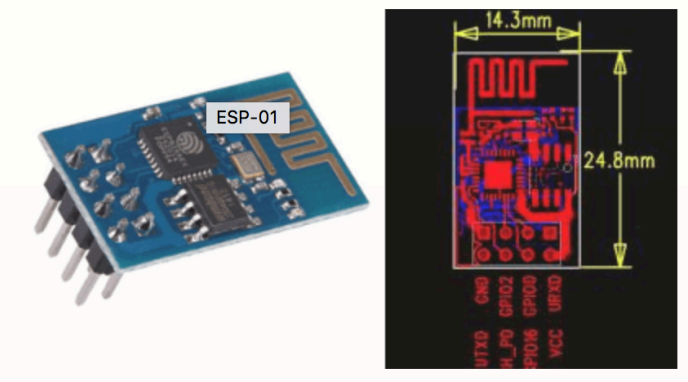


Figura 2 - ESP-01

- ESP-12E: Este módulo es el que se usará y da acceso a doce pines del ESP8266, 11 GPIO 0, 1, 2, 3, 4, 5, 12, 13, 14, 15, 16, más 1 una entrada analógica AD0, tiene antena integrada y buen alcance, por su formato es necesario un adaptador o crear un pcb.

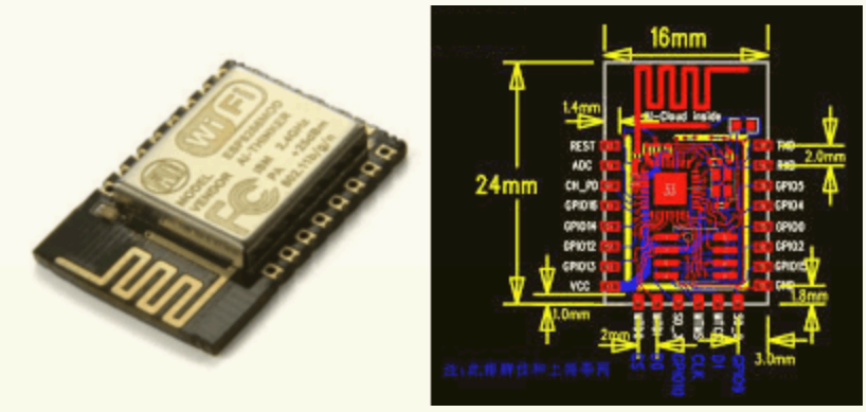


Figura 3 - ESP-12E

2.1.2 EMW3165

El primer gran competidor del ESP8266, es el llamado EMW3165.

El EMW3165 es un módulo conversor serie a wifi de bajo consumo desarrollado por MXChip. [3] Está compuesto por un microcontrolador ARM Cortex M4 de 32 bits STM32F411CE de STMicroelectronics [4], un micro mucho más potente (quizás con la capacidad de permitir tratamiento digital de señales o DSP), con más memoria RAM y FLASH, periféricos más variados (USART, I2C, SPI, ADC, DAC, PWM, TIMER...), más pines GPIO y más bajo consumo que el ESP8266.

El precio es de 1$ o 1.5$ superior al ESP8266 sobre unos 6$, más que razonable por las ventajas que proporciona.

La alimentación del módulo es a 3.3V y aunque los puertos de entrada/salida del chip son de esta tensión (máximo 3.6V), algunos de ellos presentan tolerancia a los 5V. Además, el modulo ha pasado el certificado de conformidad FCC, lo que lo convierte en una solución apta para sistemas comerciales.

La antena del módulo se encuentra implícita en la PCB pero se da soporte a la conexión de antenas externas para conseguir un mayor alcance. Así, incluyendo la zona correspondiente a la antena en PCB, las

dimensiones del módulo son de unos 3.2 cm x 1.6 cm de lado. [5]

El módulo es compatible con la librería MiCO (librería que gestiona todo el procesado del WiFi), lo que facilita el uso al desarrollador. Además, también se pueden encontrar firmwares específicos para aplicaciones típicas: UART inalámbrica, comunicación con servicios en la nube, etc.

Figura 4 - EMW3165

2.2 Programando el módulo ESP8266

El módulo viene con un firmware instalado, un SDK (kit de desarrollo de software) que se puede actualizar a las últimas versiones, descargando éstas a través de la página oficial de Espressif. Con este módulo se pueden realizar diversas tareas, desde controlar unos simples LEDS, montar un servidor WEB, controlar actuadores,… Son muchas las posibilades que ofrece.

A continuación se van a describir algunos de los métodos usados para programar el módulo ESP8266.

2.2.1 Comandos AT

El conjunto de comandos Hayes es un lenguaje desarrollado por la compañía Hayes Communications que prácticamente se convirtió en estándar abierto de comandos para configurar y parametrizar módems. Los caracteres «AT», que preceden a todos los comandos, significan «Atención», e hicieron que se conociera también a este conjunto de comandos como comandos AT.

Un aparato que implemente el conjunto de comandos Hayes se considera compatible Hayes. Parte del conjunto de comandos Hayes fue incluido por la ITU-T en el protocolo V.25ter, actual V.250. La adopción de este estándar hizo el desarrollo de controladores específicos para distintos modems. [6]

Existe una gran variedad de comandos AT para distintos fines, algunos de los que se han usa para probar este módulo son:

CONJUNTO DE COMANDOS BÁSICOS:

1. At
2. AT+GMR → Versión de Firmware
3. At+CWLAP → Ver listado de redes Wi-Fi
1. AT+CWMODE=3 → El comando AT+CWMODE nos permite asignar el modo WIFI,

   a. 1=Sta, 2=AP, 3=ambos.
4. AT+RST → Reset
5. AT+CWMODE? → Chequear el modo Wifi
6. AT+CWJAP="SSID", "Contraseña"→ Conectarse a un punto de acceso Wi-Fi.
7. AT+CWJAP? → saber a que red estamos conectados
8. AT+CIFSR → ver la IP asignada
9. AT+CWQAP --> Se desconecta de la conexión
2. 10 AT+RESTORE --> Restaura valores de fabrica

EJEMPLO ACTUANDO COMO CLIENTE TCP:

1. AT+CIPMUX=1 → Habilitar multiples conexiones
2. AT+CIPSTART=? → ayuda comando CIPSTART
3. AT+CIPSTART=0,"TCP","192.168.1.100",8080 → ID de conexión,Protocolo, Server IP, puerto
4. AT+CIPSEND=0,4 → ID y Longitud de la cadena a transmitir en Bytes
5. AT+CIPCLOSE=0 → Cierra conexión TCP NO 0

EJEMPLO COMO SERVIDOR TCP:

1. AT+CIPMUX=1 → Habilitar multiples conexiones
2. AT+CIFSR
3. AT+CIPSERVER=1,8080 → 1=crea server,Puerto por 8080
4. AT+CIPCLOSE=0 → Borra la conexión número 0 (La conexión se puede cerrar desde el cliente o desde el servidor)
5. AT+CIPSERVER=0 → 0= Borra Server (se necesita restaurar)

Estos comandos son tirados a través de programas como por ejemplo en window “terminal.exe” o “LUA uploader” disponible en múltiples plataformas. Una vez que se tiene el firmware actualizado solo es necesario seleccionar el puerto (COM?) correcto, que se ha asignado a nuestro módulo y se podrá establecer una comunicación con él.

2.2.2 Programando el ESP8266 como un Arduino

Arduino es una compañía de hardware libre, la cual desarrolla placas de desarrollo que integran un microcontrolador y un entorno de desarrollo (IDE), diseñado para facilitar el uso de la electrónica en proyectos multidisciplinarios. [7]

El software consiste en un entorno de desarrollo (IDE) basado en el entorno de Processing y lenguaje de programación basado en Wiring, así como en el cargador de arranque (bootloader) que es ejecutado en la placa. El microcontrolador de la placa se programa a través de un computador, haciendo uso de comunicación serial mediante un convertidor de niveles RS-232 a TTL serial.

Existe un paquete que se puede instalar a este software que permite hacer uso de este IDE de arduino para programar el ESP8266. Basta con añadir en las preferencias del programa la url “http://arduino.esp8266.com/stable/package_esp8266com_index.json” y posteriormente en en herramientas donde se elige la placa con la que se trabaja, instalar desde el gestor de tarjetas el paquete correspondiente al ESP8266.

Una vez instalado este paquete se pondrá la siguiente configuración:

1. Herramientas → Programador → esptool (Es una herramienta para crear archivos de firmware para el ESP8266 y flashear en la memoria del chip a través del puerto serie ese firmware).
2. Se comprueba que la placa está seleccionada: Placa → Generic ESP8266 board
3. Comprobar que se selecciona el puerto adecuado: Puerto → COM?

Cuando se finalicen estos pasos el IDE de arduino estará listo para empezar a probar ejemplos en el módulo ESP8266. Se puede empezar probando ejemplos sencillos como el "blinky led" que está dentro de "Archivo → Ejemplos → Basics → Blink" , también se puede implementar un servidor web para encender y apagar un led ("Archivo → Ejemplos → ESP8266WiFi → WiFiWebServer"). Existen diversos ejemplos que pueden resultar útiles para hacer pruebas o para modificarlos como se desee en la aplicación concreta.

Aquí el ejemplo del "blink":

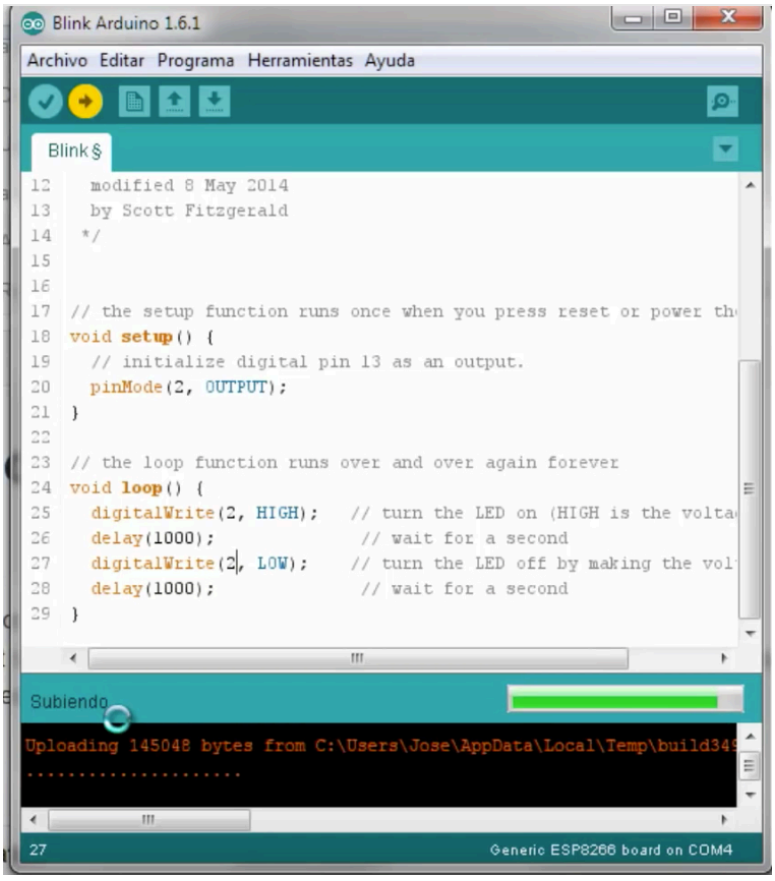


Figura 5 - Arduino ejemplo "blink"

2.2.3 NodeMCU

Es un proyecto de código abierto, basado en el ESP8266 WiFi SOC de Espressif. Básicamente es un firmware para este módulo basado en Lua, ha cogido el SDK de Espressif y lo ha modificado, el código está escrito en su mayor parte C.

Con este firmware es con el que se va a trabajar en lo capítulos posteriores y sobre el que se centra este trabajo.

Para flashear este firmware en el módulo se usa el programa "esptool" o el "NODEMCU Flasher" ambos proporcionados en la página de NodemMCU. [8] En esta página viene el código fuente de este firmware así

como algún ejemplo, configuración de los pines para poner el ESP8266 en modo flash, los módulos que actualmente están integrados, …

Para programar el dispositivo con NodeMCU de Lua, se usa el programa "ESplorer", el cual se descarga de la url anterior. Con este programa se le pasan los script que se hayan escrito en Lua. Se puede hacer multitud de aplicaciones con este firmware algunos ejemplos vienen en la página web oficial de NodeMCU y van desde el blinky, encender y apagar un LED, conectarse a una red WiFi, crear un servidor web y apagar y encender un led desde la página web (como el ejemplo de arduino pero ahora usando código Lua),… hasta cualquier aplicación o ejemplo concreto que se desee probar. [9]

2.2.3.1 LUA

Lua es un lenguaje potente, eficiente, ligero, integrable,… Es compatible con la programación procedimental, la programación orientada a objetos, programación funcional, programación basada en datos, y la descripción de los datos.

Combina la sintaxis de procedimiento sencillo con poderosas construcciones de descripción de datos basados en matrices asociativas y semánticas extensibles. Lua se escribe de forma dinámica, se ejecuta mediante la interpretación de código de bytes con una máquina virtual basada en registros, y tiene la gestión de memoria automática con la recolección de basura, lo que es ideal para la configuración, scripting, y prototipado rápido. [10]

Lua está diseñado, implementado y mantenido por un equipo en la PUC-Rio , la Pontificia Universidad Católica de Río de Janeiro en Brasil. Lua nació y se crió en Tecgraf , anteriormente el Grupo de Gráficos Tecnología de Computadores de la PUC-Rio. Lua se encuentra ahora en LabLua , un laboratorio del Departamento de Ciencias de la Computación de la PUC-Rio.

Es un lenguaje robusto, se ha utilizado en muchas aplicaciones industriales (por ejemplo, de Adobe Photoshop Lightroom), con énfasis en los sistemas integrados (por ejemplo, el Ginga middleware para televisión digital en Brasil) y juegos (por ejemplo, World of Warcraft y Angry Birds). Lua es actualmente el lenguaje de secuencias de comandos que lleva en los juegos . Lua tiene un buen manual de referencia y hay varios libros sobre el tema . Varias versiones de Lua han sido lanzadas y se utilizan en aplicaciones reales desde su creación en 1993.

Es rápido, tiene una merecida reputación por su rendimiento. Varios referentes muestran Lua como el idioma más rápido en el ámbito de los lenguajes de script interpretado. Si necesita más velocidad, puede probar a usar LuaJIT, un compilador en tiempo de ejecución. [10]

También es portable, se distribuye en un paquete pequeño y funciona en todas las plataformas que tienen un compilador de C estándar. Lua se ejecuta en todas las versiones de Unix y Windows, en los dispositivos móviles (con Android, iOS, BREW, Symbian, Windows Phone), en microprocesadores embebidos (como ARM y Rabbit, para aplicaciones como Lego Mindstorms), en unidades centrales de IBM, etc.

Su tamaño reducido hace que se puede integrar fácilmente en su aplicación. Lua tiene una API simple y bien documentada que permite una fuerte integración con el código escrito en otros idiomas. Es fácil de extender Lua con bibliotecas escritas en otros idiomas. También es fácil de extender los programas escritos en otros idiomas con Lua. Lua se ha utilizado para extender los programas escritos no sólo en C y C ++, sino también en Java, C #, Smalltalk, Fortran, Ada, Erlang, e incluso en otros lenguajes de script, como Perl y Ruby.

Un concepto fundamental en el diseño de Lua es proporcionar meta-mecanismos para la implementación de funciones, en lugar de proporcionar una serie de características directamente en la lengua. Por ejemplo, aunque Lua no es un lenguaje orientado a objetos puro, proporciona meta-mecanismos de implementación de clases y herencia.

Lua es software libre de código abierto, distribuido bajo una licencia muy liberal (la licencia MIT). Puede ser utilizado para cualquier propósito, incluyendo fines comerciales, sin ningún costo. Sólo descargar y usarlo. [11]

2.3 WPS

WPS (Wi-Fi Protected Setup) es un estándar de 2007, promovido por la Wi-Fi Alliance para facilitar la creación de redes WLAN. En otras palabras, WPS no es un mecanismo de seguridad de por sí, se trata de la definición de diversos mecanismos para facilitar la configuración de una red WLAN segura con WPA2, pensados para minimizar la intervención del usuario en entornos domésticos o pequeñas oficinas otros(SOHO). Concretamente, WPS define los mecanismos a través de los cuales los diferentes dispositivos de la red obtienen las credenciales (SSID y PSK) necesarias para iniciar el proceso de autenticación.

WPS define una arquitectura con tres elementos con roles diferentes:

- Registrar (matriculador): dispositivo con la autoridad de generar o revocar las credenciales en la red. Tanto un AP como cualquier otra estación o PC de la red pueden tener este rol. Puede haber más de un Registrar en una red.
- Enrollee (matriculado): dispositivo que solicita el acceso a la red WLAN.
- Authenticator (autenticador): AP funcionando de proxy entre el Registrar y el Enrollee.

WPS contempla cuatro tipos de configuraciones diferentes para el intercambio de credenciales, PIN (Personal Identification Number), PBC (Push Button Configuration), NFC (Near Field Communications) y USB (Universal Serial Bus):

1. PIN: tiene que existir un PIN asignado a cada elemento que vaya a asociarse a la red. Este PIN tiene que ser conocido tanto por el Registrar, como por el usuario (Enrollee). Es necesaria la existencia de una interfaz (por ejemplo pantalla y teclado) para que el usuario pueda introducir el mencionado PIN.
2. PBC: la generación y el intercambio de credenciales son desencadenados a partir que el usuario presiona un botón (físico o virtual) en el AP (o en otro elemento Registrar) y otro en el dispositivo. Notar que en el corto lapso de tiempo entre que se presiona el botón en el AP y se presiona en el dispositivo, cualquier otra estación próxima puede ganar acceso a la red.
3. NFC: intercambio de credenciales a través de comunicación NFC. La tecnología NFC, basada en RFID permite la comunicación sin hilos entre dispositivos próximos (0 - 20 cm). Entonces, el dispositivo Enrollee se tiene que situar al lado del Registrar para desencadenar la autenticación. De esta manera, cualquier usuario que tenga acceso físico al Registrar, puede obtener credenciales válidas.
4. USB: con este método, las credenciales se transfieren mediante un dispositivo de memoria flash (por ejemplo pendrive) desde el Registrar al Enrollee.

Los métodos PBC, NFC y USB pueden usarse para configurar dispositivos sin pantalla ni teclado (impresoras, webcams, etc.), pero aunque el estándar contempla NFC y USB, todavía no se certifican estos mecanismos. Actualmente sólo el método PIN es obligatorio en todas las estaciones para obtener la certificación WPS; PBC es obligatorio sólo en APs.

Existe un fallo de seguridad que Stefan Viehböck descubrió en diciembre del 2011. Afecta a routers inalámbricos que tienen la función WPS (también llamada QSS), que en dispositivos actuales se encuentra habilitada en forma preestablecida. El fallo permite a un atacante recuperar el PIN WPS y, con él, la clave pre-compartida de la red WPA/WPA2 usando ataques de fuerza bruta en pocas horas. Los usuarios deben deshabilitar la función WPS como solución temporal. Es posible que en algunos dispositivos no pueda realizarse este procedimiento. [12]

3 ENTORNO DE DESARROLLO

Un entorno de desarrollo no es más que una aplicación informática que proporciona servicios integrales para facilitar al desarrollador o programador el desarrollo de software. Normalmente consiste en un editor de código fuente, herramientas de construcción automáticas y un depurador. La mayoría de los IDE tienen auto-completado inteligente de código. Algunos IDE contienen un compilador, un intérprete, o ambos, tales como NetBeans y Eclipse.

Uno de los propósitos de los IDE es reducir la configuración necesaria para reconstruir múltiples utilidades de desarrollo, en vez de proveer el mismo set de servicios como una unidad cohesiva. Reduciendo ese tiempo de ajustes, se puede incrementar la productividad de desarrollo, en casos donde aprender a usar un IDE es más rápido que integrar manualmente todas las herramientas por separado.

Una mejor integración de todos los procesos de desarrollo hace posible mejorar la productividad en general, más que únicamente ayudando con los ajustes de configuración. Por ejemplo, el código puede ser continuamente armado, mientras es editado, previendo retroalimentación instantánea, como cuando hay errores de sintaxis. Esto puede ayudar a aprender un nuevo lenguaje de programación de una manera más rápida, así como sus librerías asociadas.

Primeramente se usará eclipse para modificar el código, generar los binarios y flashear algunos ejemplos, después una vez se hayan probado se pasará a usar una herramienta llamada “Docker”, que es la que se usará para compilar el firmware de NodeMCU.

3.1 Eclipse

Para el desarrollo del proyecto se ha escogido el IDE de Eclipse. Eclipse es una plataforma de software compuesto por un conjunto de herramientas de programación de código abierto multiplataforma para desarrollar lo que el proyecto llama "Aplicaciones de Cliente Enriquecido", opuesto a las aplicaciones "Cliente-liviano" basadas en navegadores. Esta plataforma, típicamente ha sido usada para desarrollar entornos de desarrollo integrados, como el IDE de Java llamado Java Development Toolkit (JDT) y el compilador (ECJ) que se entrega como parte de Eclipse (y que son usados también para desarrollar el mismo Eclipse). Sin embargo, también se puede usar para otros tipos de aplicaciones cliente, como BitTorrent o Azureus.

Eclipse fue desarrollado originalmente por IBM como el sucesor de su familia de herramientas para VisualAge. Eclipse es ahora desarrollado por la Fundación Eclipse, una organización independiente sin ánimo de lucro que fomenta una comunidad de código abierto y un conjunto de productos complementarios, capacidades y servicios.

Eclipse fue liberado originalmente bajo la Common Public License, pero después fue re-licenciado bajo la Eclipse Public License. La Free Software Foundation ha dicho que ambas licencias son licencias de software libre, pero son incompatibles con Licencia pública general de GNU (GNU GPL).

La base para Eclipse es la Plataforma de cliente enriquecido (del inglés Rich Client Platform RCP). Los siguientes componentes constituyen la plataforma de cliente enriquecido:

- Plataforma principal - inicio de Eclipse, ejecución de plugins.
- OSGi - una plataforma para bundling estándar.
- El Standard Widget Toolkit (SWT) - Un widget toolkit portable.
- JFace - manejo de archivos, manejo de texto, editores de texto.
- El Workbench de Eclipse - vistas, editores, perspectivas, asistentes.

El entorno de desarrollo integrado (IDE) de Eclipse emplea módulos (en inglés plug-in) para proporcionar toda su funcionalidad al frente de la plataforma de cliente enriquecido, a diferencia de otros entornos monolíticos donde las funcionalidades están todas incluidas, las necesite el usuario o no. Este mecanismo de módulos es una plataforma ligera para componentes de software. Adicionalmente a permitirle a Eclipse extenderse usando otros lenguajes de programación como son C/C++ y Python, permite a Eclipse trabajar con lenguajes para procesado de texto como LaTeX, aplicaciones en red como Telnet y Sistema de gestión de base de datos. La arquitectura plugin permite escribir cualquier extensión deseada en el ambiente, como sería Gestión de la configuración. Se provee soporte para Java y CVS en el SDK de Eclipse. Y no tiene por qué ser usado únicamente con estos lenguajes, ya que soporta otros lenguajes de programación.

La definición que da el proyecto Eclipse acerca de su software es: "una especie de herramienta universal - un IDE abierto y extensible para todo y nada en particular".

En cuanto a las aplicaciones clientes, Eclipse provee al programador con frameworks muy ricos para el desarrollo de aplicaciones gráficas, definición y manipulación de modelos de software, aplicaciones web, etc.

El SDK de Eclipse incluye las herramientas de desarrollo de Java, ofreciendo un IDE con un compilador de Java interno y un modelo completo de los archivos fuente de Java. Esto permite técnicas avanzadas de refactorización y análisis de código. Mediante diversos plugins estas herramientas están también disponibles para otros lenguajes como C/C++ (Eclipse CDT) y en la medida de lo posible para lenguajes de script no tipados como PHP o Javascript. El IDE también hace uso de un espacio de trabajo, en este caso un grupo de metadatos en un espacio para archivos planos, permitiendo modificaciones externas a los archivos.

Eclipse dispone de un Editor de texto con un analizador sintáctico. La compilación es en tiempo real. Tiene pruebas unitarias con JUnit, control de versiones con CVS, integración con Ant, asistentes (wizards) para creación de proyectos, clases, tests, etc., y refactorización. [13]

Asimismo, a través de "plugins" libremente disponibles es posible añadir más funcionalidades.

3.1.1 Configuración de eclipse

La versión de eclipse que se ha utilizado es Eclipse Marte x86 para Windows. Para crear un entorno en el que se pudiera trabajar con el ESP8266, programarlo, probar ejemplos, flashearlo,… Se han llevado a cabo los siguientes pasos: [14]

1- Descargar e instalar el “Unofficial Development Kit for Espressif ESP8266” se llama “Espressif-ESP8266-DevKit-v2.0.9-x86.exe” (contiene códigos de ejemplo para el ESP8266 con los que podemos probar el dispositivo y modificarlos).

2- Descargar e instalar el “Java Runtime x86 (jre-7uXX-windows-i586.exe)” o la versión actualizada “jre-8xx…”. A partir de la “jre-7xx…” son todas válidas.

3- Descargar e instalar Eclipse Marte x86 para desarrollar en C ++ (eclipse-cpp-mars-R-win32.zip). Descomprimir el archivo a la raíz de la unidad C.

4- Descargar e instalar MinGW (Minimalist GNU for Window). Ejecutar "mingw-get-setup.exe", en el proceso de instalación seleccionar sin interfaz gráfica de usuario (GUI), es decir, desactivar la opción "... also install support for the graphical user interface".

MinGW (Minimalist GNU for Windows), anteriormente conocido como MinGW32, es una implementación de los compiladores GCC para la plataforma Win32, que permite migrar la capacidad de este compilador en entornos Windows. Es un fork de Cygwin en su versión 1.3.3. Además MinGW incluye un conjunto de la API de Win32, permitiendo un desarrollo de aplicaciones nativas para esa plataforma, pudiendo generar ejecutables y bibliotecas usando la API de Windows.

5- Para automatizar la instalación de los módulos adicionales para MinGW usar el script "install-mingw-package ". Ejecutar el archivo "install-mingw-package" y se instalarán los módulos básicos para MinGW, la instalación debe proceder sin error.

El código de este fichero es:

```
@echo off
title Installing additional packages for MinGW
cls
echo.
echo Installing additional packages for MinGW
echo.
at > nul
if not %ERRORLEVEL% EQU 0 (
      echo Error: To install packages need admin rights.
      echo.
      echo Run the script as an administrator.
      echo.
      echo To run as administrator, click on the file install.cmd
the right mouse
      echo button and select 'Run as administrator'
      echo.
      echo Press Enter to exit.
      pause >nul
      goto :eof
)
echo Press Ctrl-C to cancel, Enter to continue
pause >nul
cd /d %~dp0
echo.

if exist "C:\MinGW" (
  if not exist "C:\MinGW\var\cache\mingw-get" (
    mkdir C:\MinGW\var\cache\mingw-get
```

```
)
echo Copying mingw-cache-packages.zip to
C:\MinGW\var\cache\mingw-get
copy /Y mingw-cache-packages.zip C:\MinGW\var\cache\mingw-get\
copy /Y unzip.exe C:\MinGW\var\cache\mingw-get
C:
cd C:\MinGW\var\cache\mingw-get
if exist mingw-cache-packages.zip (
  echo Extract mingw-cache-packages.zip...
  unzip.exe -o mingw-cache-packages.zip
  del mingw-cache-packages.zip
  del unzip.exe
)
C:
cd C:\MinGW\bin
if exist mingw-get.exe (
  echo Installing mingw packages...
  mingw-get install mingw32-base
  mingw-get install mingw32-mgwport
  mingw-get install mingw32-pdcurses
  mingw-get install mingw32-make
  mingw-get install mingw32-autoconf
  mingw-get install mingw32-automake
  mingw-get install mingw32-gdb
  mingw-get install gcc
  mingw-get install gcc-c++
  mingw-get install libz
  mingw-get install bzip2
  mingw-get install msys-base
  rem mingw-get install msys-coreutils
  mingw-get install msys-coreutils-ext
  mingw-get install msys-gcc-bin
  mingw-get install msys-wget-bin
  mingw-get install msys-m4
  mingw-get install msys-bison-bin
  mingw-get install msys-flex-bin
  mingw-get install msys-gawk
  rem mingw-get install msys-sed
  mingw-get install msys-autoconf
```

```
        mingw-get install msys-automake
        mingw-get install msys-mktemp
        rem mingw-get install msys-patch
        mingw-get install msys-libtool
        echo.
        echo Installing additional packages for MinGW complete.
        echo.
        echo Press Enter to exit the installation wizard.
        echo.
        pause >nul
      ) else (
        echo.
        echo ERROR! mingw-get.exe not found.
        echo.
        echo Press Enter to exit.
        echo.
        pause >nul
        goto :eof
      )
    ) else (
      echo.
      echo ERROR! MinGW not found.
      echo.
      echo Press Enter to exit.
      echo.
    )
```

6- Iniciar el Eclipse de Luna desde el directorio c: \ eclipse \ eclipse.exe

7- En Eclipse, seleccione Archivo → Importar → General → Proyecto en área de trabajo existente, en el directorio raíz Seleccionar línea, seleccione el directorio C: \ Espressif \ ejemplos y proyectos de trabajo de importación.

Una vez realizados todos estos pasos se tendrá un entorno en el que poder trabajar y probar los ejemplos para el ESP8266 e incluso flasearlos directamente desde Eclipse.

3.2 Primeras pruebas (usando Eclipse)

En esta sección se van a explicar algunos de los ejemplos del kit de spressif que se han probado en el módulo ESP8266 para comprobar su correcto funcionamiento y entender como funciona el módulo, la forma de programación,… Todos estos ejemplos se han creado y flasheado usando Eclipse con la configuración descrita anteriormente.

3.2.1 Modo flash del ESP8266

Para flashear el módulo basta con ponerlo en modo flash y cuando se conecte el ordenador reconoce que esta listo para grabarle el firmware que se desee. Para poner el ESP8266 en modo flash se utiliza la siguiente configuración de los pines:

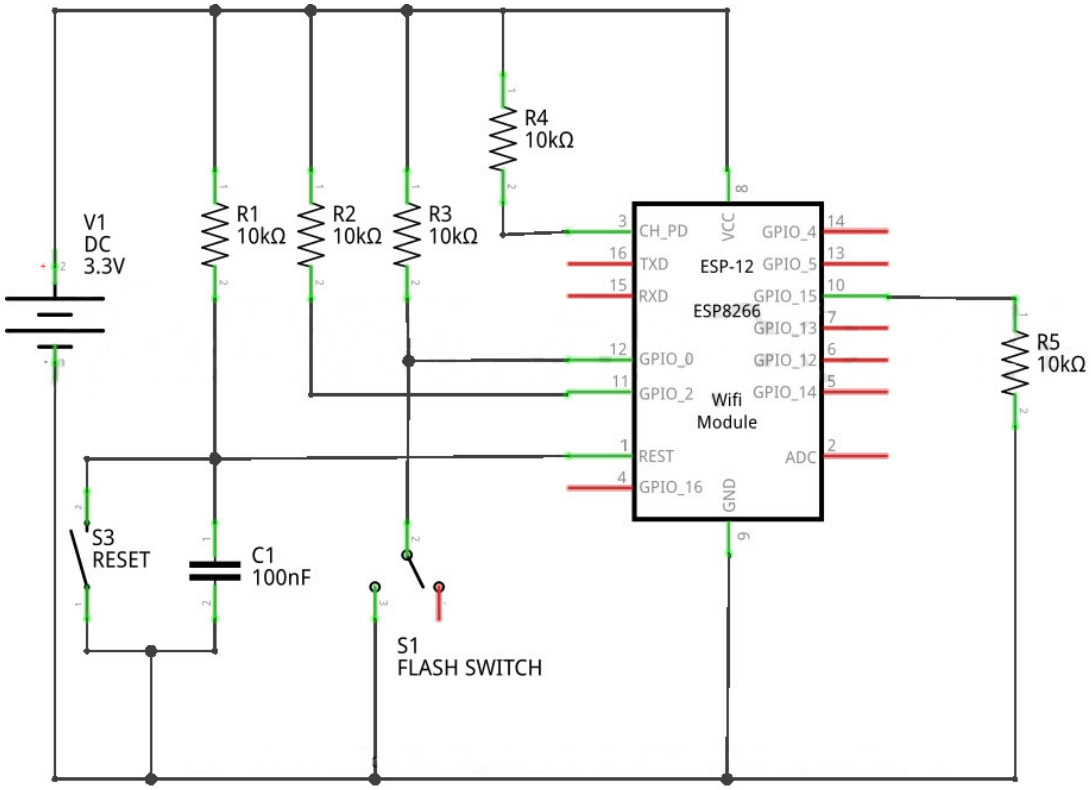


Figura 6 - Modo Flash ESP8266

Este es un ejemplo de configuración de los pines para modo flash. No es necesario poner todas las resistencias de pull-up para conectar a Vcc o todos los interruptores que vienen aquí, pero si se recomienda su uso. Lo más importante a tener en cuenta que si se observa en el esquema es:

- CH_PD, y GPIO_2 siempre en HIGH.
- GPIO_15 siempre a LOW
- GPIO_0 a HIGH para el funcionamiento normal del chip o a GND para cuando necesitemos flashear el chip. En este caso el modo normal se activa cuando el interruptor S1 esta abierto o modo flasheo cuando S1 esta cerrado.
- REST a HIGH en modo normal y cuando se quiere resetear el módulo a LOW. En este caso cuando se pulsa el pulsador S3 se resetea (reinicializa) el chip. Necesario si queremos resetear el chip en modo normal o resetearlo en modo flasheo justo después de cerrar el interruptor S1.

3.2.2 Hello World

El primer ejemplo que se ha probado ha sido el ya conocido “Hola Mundo”, el código en c usado es:

```
#include <ets_sys.h>
#include <osapi.h>
#include <os_type.h>
```

```
#include <gpio.h>
#include "driver/uart.h"

#define DELAY 1000 /* milliseconds */

LOCAL os_timer_t hello_timer;
extern int ets_uart_printf(const char *fmt, ...);

LOCAL void ICACHE_FLASH_ATTR hello_cb(void *arg)
{
      ets_uart_printf("Hello World!\r\n");
}

void user_rf_pre_init(void)
{
}

void user_init(void)
{
      // Configure the UART
      uart_init(BIT_RATE_115200, BIT_RATE_115200);
      // Set up a timer to send the message
      // os_timer_disarm(ETSTimer *ptimer)
      os_timer_disarm(&hello_timer);
      // os_timer_setfn(ETSTimer *ptimer, ETSTimerFunc *pfunction, void
*parg)
      os_timer_setfn(&hello_timer, (os_timer_func_t *)hello_cb, (void
*)0);
      // void os_timer_arm(ETSTimer *ptimer,uint32_t milliseconds, bool
repeat_flag)
      os_timer_arm(&hello_timer, DELAY, 1);
}
```

Como se observa es una programación por eventos, en la que primero se ejecuta la rutina use_init y posteriormente los distintos eventos que sean llamados desde esta. En este caso la función “hello_cb”.

Al compilar este código, flashearlo en el módulo y conectarse por el puerto serie usando el programa “LUA Uploader” por ejemplo, lo que se consigue es que se muestre por pantalla cada 1000 milisegundos “Hello Wolrd”.

3.2.3 Blinky LED

Otro ejemplo bastante típico es el parpadeo de un led, se ha programado en c usando eclipse, al igual que el anterior también se ha compilado con eclipse y flasheado usando el “esptool” desde eclipse. El código de este ejemplo es:

```
#include <ets_sys.h>
#include <osapi.h>
#include <gpio.h>

// see eagle_soc.h for these definitions
#define LED_GPIO 4
#define LED_GPIO_MUX PERIPHS_IO_MUX_GPIO4_U
#define LED_GPIO_FUNC FUNC_GPIO4
```

```
#define DELAY 500000 /* microseconds */

extern void ets_wdt_enable (void);
extern void ets_wdt_disable (void);

void user_rf_pre_init(void)
{
}

void user_init(void)
{
      uint8_t state=0;
      ets_wdt_enable();
      ets_wdt_disable();
      // Configure pin as a GPIO
      PIN_FUNC_SELECT(LED_GPIO_MUX, LED_GPIO_FUNC);
      for(;;)
      {
            GPIO_OUTPUT_SET(LED_GPIO, state);
            os_delay_us(DELAY);
            state ^=1;
      }
}
```

Este código configura el pin 4 como un GPIO para poder conectar el led a éste e inicializa este pin a nivel bajo 0 y con el bucle for se entra en un bucle infinito en el que va cambiando el estado del pin entre LOW (0 → apagado) y HIGH (1 → encendido) con un retardo de 500000 milisegundos.

3.3 Docker

Es una imagen configurada que te proporcionan los desarrolladores del NodeMCU, que te permite usando la maquina virtual "Virtual Box" con esta imagen compilar el código fuente del NodeMCU sin necesidad de configurar tu manualmente el entorno de desarrollo para Mac.

Para instalar "docker" hay que ir a la página oficial y descargarse el instalador para Windows, Mac o Linux, y seguir los pasos que se detallan. En este caso se ha usado "Docker Toolbox". [15] Aparece la siguiente ventana:

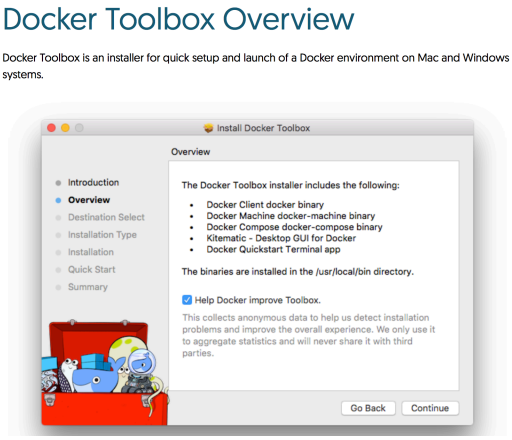


Figura 7 - Instalación "Docker Toolbox" para Mac

Solo hay que seguir los pasos, este paquete de “Docker Toolbox” instala el “Docker Quickstart Terminal”, “Kitematic” y “VirtualBox”. Se usará el “Docker Quickstart Terminal”, al abrirlo aparece un terminal como este con la imagen corriendo, para comprobar si docker está corriendo correctamente se usará el comando:

```
$> docker run hello-world
```

Si se ha iniciado todo correctamente debería aparecer una respuesta como la de la imagen:

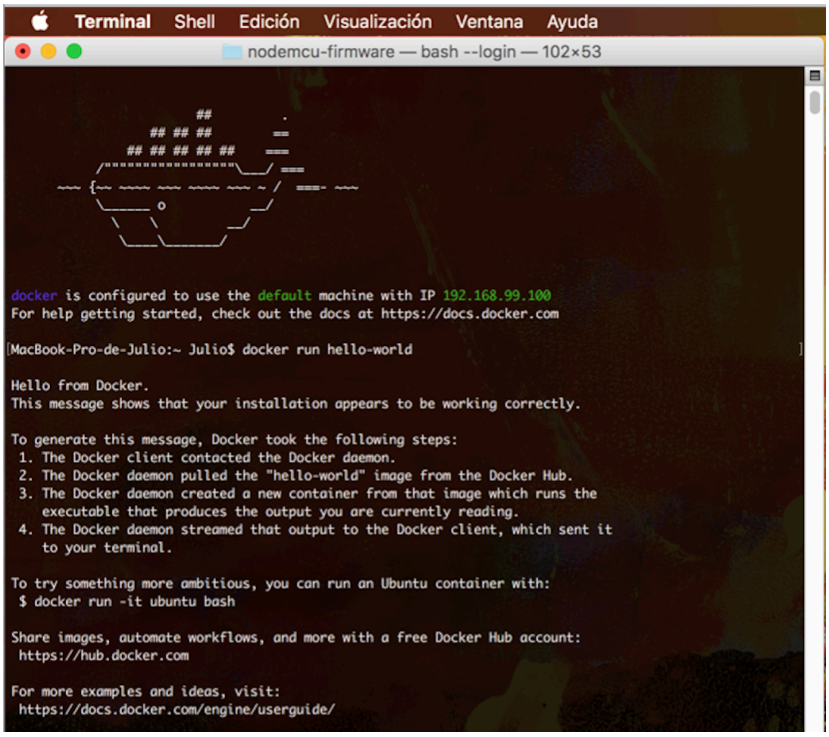


Figura 8 - Corriendo "Docker"

3.3.1 Compilar el firmware NodeMCU

Una vez llegados a este punto tenemos listo nuestro entorno para compilar el código de NodeMCU. [16] Para ello se puede descargar como zip el código de la página o clonar el repositorio directamente desde github con el comando:

```
$> git clone https://github.com/nodemcu/nodemcu-firmware.git
```

Se deberá clonar en mac en `"/Users/<user>"` o descomprimir el zip si lo hemos hecho de esa forma en esta ruta también.

Ahora cambiamos al directorio del firmware de NodeMCU y para compilarnos ejecutar:

```
$> docker run --rm -ti -v `pwd`:/opt/nodemcu-firmware marcelstoer/nodemcu-build
```

Deberá compilar sin errores:

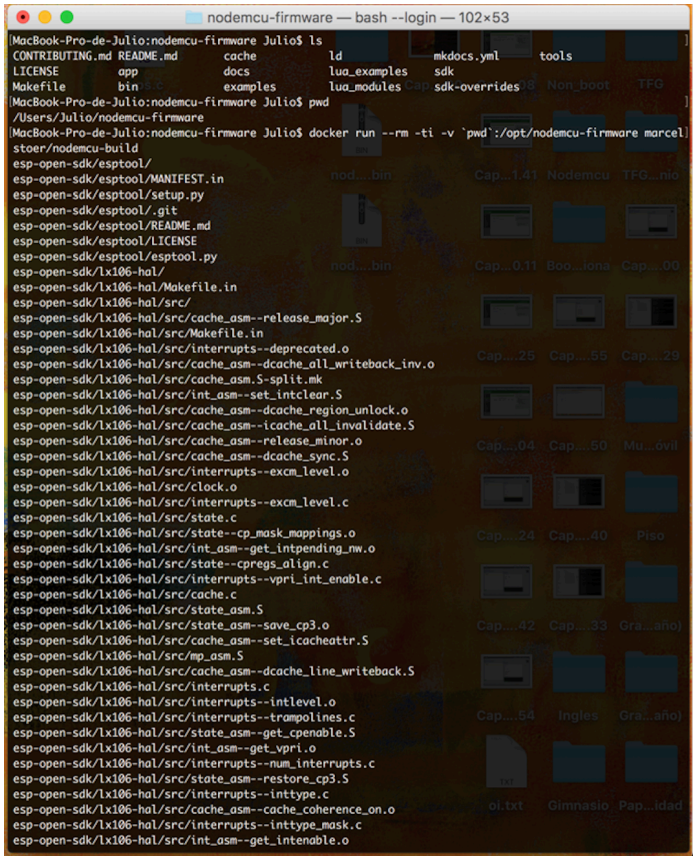


Figura 9 - Compilando NodeMCU

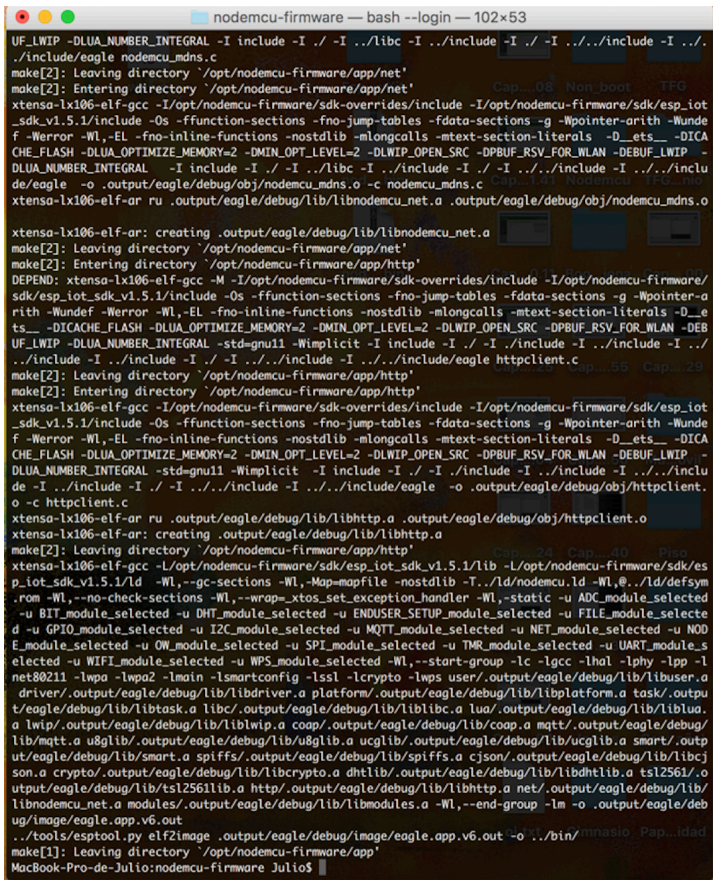


Figura 10 - Compilación terminada

3.3.2 Flashear el firmware de NodeMCU

Existen varias herramientas para actualizar y flashear el firmware del ESP8266 como son "esptool" o el "NODEMCU Flasher".

El "esptool" es una sencilla herramienta que permite comunicarse con el gestor de arranque ROM del ESP8266. Se pretende que sea un forma sencilla de flashear el módulo e independiente de la plataforma. Actualmente las plataformas compatible son OS X, Linux, Windows,… Cualquiera en la que Python funcione. Esta herramienta tiene muchas opciones para cambiar el modo en el que se graba (QIO, DUO,..), para ajustar los baudios (9600, 115200,..),… Pero para flashear el binario generado por el docker basta con tirar el siguiente comando únicamente seleccionando el puerto y la dirección en la que se desea grabar:

```
$> ./esptool.py --port <USB-port-with-ESP8266> write_flash 0x00000
<NodeMCU-firmware-directory>/bin/nodemcu_[integer|float]_<Git-branch>.bin
```

También se ha probado la otra herramienta "NODEMCU Flasher", hasta ahora solamente disponible para Windows. Esta es la interfaz de la aplicación:

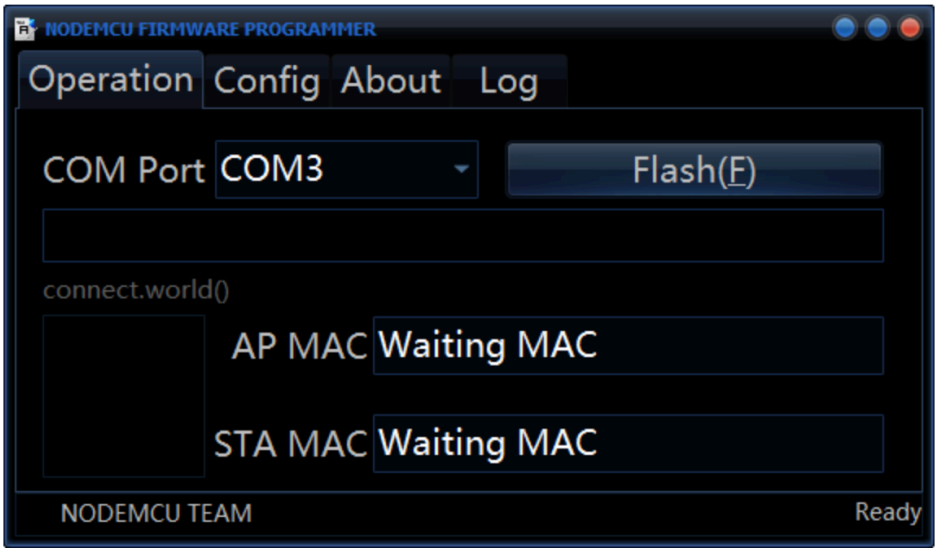


Figura 11 - NODEMCU Flasher 1/4

Se le da a flash y esperamos un momento:

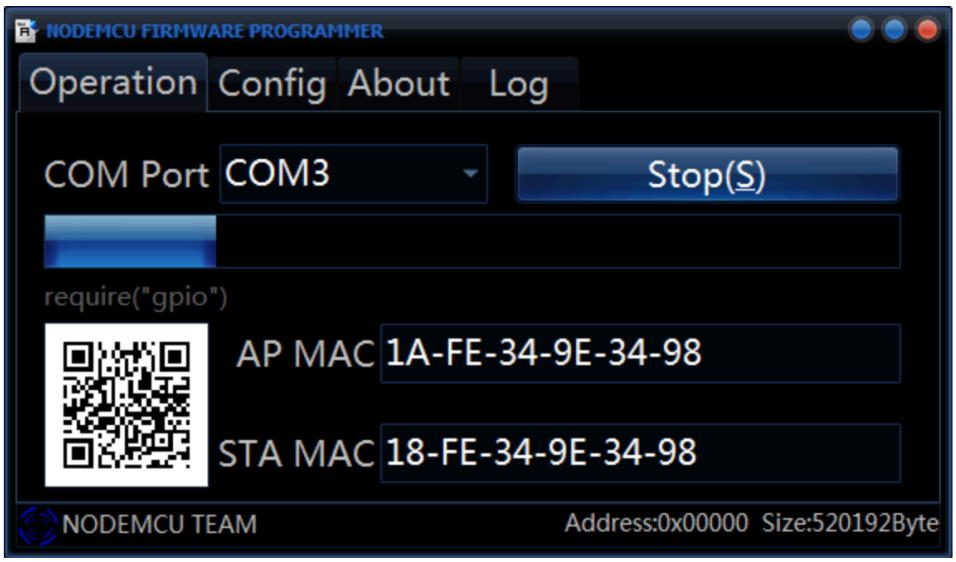


Figura 12 - NODEMCU Flasher 2/4

Aparece el tick verde en esquina inferior izquierda indicando que el programa se grabó con éxito:

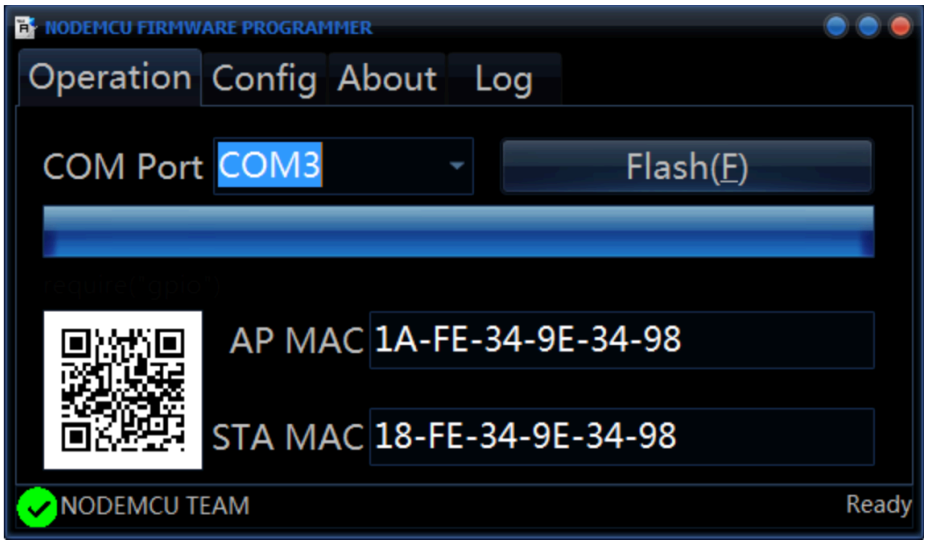


Figura 13 - NODEMCU Flasher 3/4

También se puede ajustar el firmware que se desea grabar:

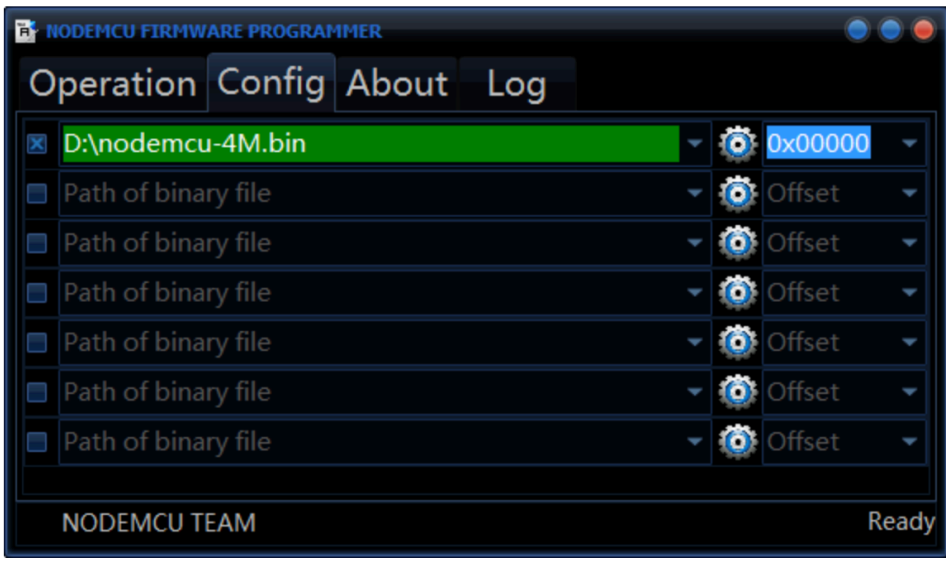


Figura 14 - NODEMCU Flasher 4/4

En la pestaña “config” se puede ajustar el modo, los baudios,…

Se ha usado el esptool por su disponibilidad en múltiples plataformas y porque tarda menos tiempo en grabar el firmware.

3.4 Ejemplos con el NodeMCU

3.4.1 Apagar y encender un LED

Una vez compilado y flasheado el NodeMCU en el ESP8266 vamos a escribir algunos scripts para probar este firmware. El primero ejemplo que se va a ver es como controlar un LED, se comprobará como ahora es mucho más sencillo y basta con tirar 5 o 6 comandos únicamente. Estos scripts se han ejecutado usando el programa “ESPlorer”.

El script usado es:

```
pin = 4
gpio.mode(pin, gpio.OUTPUT)
gpio.write(pin, gpio.HIGH)
print(gpio.read(pin))
```

Con este script se configura el pin 4 como un GPIO de salida y se pone a nivel alto (HIGH), después se lee el estado del pin. Basta como con cambiar la tercera línea por "gpio.write(pin, gpio.LOW)" para apagar el led.

3.4.2 Conectarse a una red WiFi

Se ha visto la simplicidad del script anterior para controlar un led, pues bien usando Lua también es muy sencillo conectarse a una red Wi-Fi, se muestra en el siguiente script:

```
ip = wifi.sta.getip()
print(ip)
--nil (aún no estamos conectados)
```

Nos muestra la dirección IP que tenemos asignada. Aún no estamos conectados luego mostrará nil.

```
wifi.setmode(wifi.STATION)
wifi.sta.config("SSID", "password")
```

Configuramos el modo WiFi como modo estación (había dos, estación nosotros nos conectamos, y punto de acceso se conectan a nuestro módulo. Y se conecta a la red WiFi.

```
ip = wifi.sta.getip()
print(ip)
--192.168.18.110 (se muestra la IP asignada)
```

Nos muestra la dirección IP que tenemos asignada.

3.4.3 Crear un servidor web para apagar y encender un LED

Para este ejemplo se ha usado el script llamado "init.lua" cuyo código es:

```
wifi.setmode(wifi.STATION)
wifi.sta.config("YOUR_NETWORK_NAME","YOUR_NETWORK_PASSWORD")
print(wifi.sta.getip())
led1 = 3
led2 = 4
gpio.mode(led1, gpio.OUTPUT)
```

```
gpio.mode(led2, gpio.OUTPUT)
srv=net.createServer(net.TCP)
srv:listen(80,function(conn)
    conn:on("receive", function(client,request)
        local buf = "";
        local _, _, method, path, vars = string.find(request, "([A-
Z]+) (.+)?(.+) HTTP");
        if(method == nil)then
            _, _, method, path = string.find(request, "([A-Z]+) (.+)
HTTP");
        end
        local _GET =
        if (vars ~= nil)then
            for k, v in string.gmatch(vars, "(%w+)=(%w+)&*") do
                _GET[k] = v
            end
        end
        buf = buf.."<h1> ESP8266 Web Server</h1>";
        buf = buf.."<p>GPIO0 <a
href=\"?pin=ON1\"><button>ON</button></a> <a
href=\"?pin=OFF1\"><button>OFF</button></a></p>";
        buf = buf.."<p>GPIO2 <a
href=\"?pin=ON2\"><button>ON</button></a> <a
href=\"?pin=OFF2\"><button>OFF</button></a></p>";
        local _on,_off = "",""
        if(_GET.pin == "ON1")then
              gpio.write(led1, gpio.HIGH);
        elseif(_GET.pin == "OFF1")then
              gpio.write(led1, gpio.LOW);
        elseif(_GET.pin == "ON2")then
              gpio.write(led2, gpio.HIGH);
        elseif(_GET.pin == "OFF2")then
              gpio.write(led2, gpio.LOW);
        end
        client:send(buf);
        client:close();
        collectgarbage();
    end)
```

Las primeras líneas del código sirven para conectarnos a una red Wi-Fi como se ha visto en el ejemplo anterior. Una vez que tenemos acceso a internet configuramos los pines en este caso 3 y 4 como GPIO de salida y se crea un servidor web que escucha en el puerto 80.
Posteriormente se crea una página web con dos botones uno de encendido (ON) y otro de apagado (OFF) para cada LED.
Ahora basta con tirar el comando “print(wifi.sta.getip())” para ver a que IP ha sido asignada al ESP8266 y accediendo a esa IP desde un navegador se accede a la página web creada y se pueden controlar los LED.

4 REALIZACIÓN Y DESARROLLO

En este capítulo se va a crear una ejemplo para comprobar que el ESP8266 puede usar el protocolo WPS para conectarse a una red WiFi. Posteriormente tras comprobar que esto es posible se explicará como se crea un módulo nuevo en NodeMCU con sus funciones.

Una vez explicado todo esto se creará el nuevo módulo WPS dentro del NodeMCU, detallando todos los pasos y modificaciones necesarias en el código.

4.1 WPS para el ESP8266

Lo primero que se va a hacer es actualizar el SDK de Espressif a la última versión, la "esp_iot_sdk_v1.5.1". Porque la librería que se va a usar, la "libwps.a" se incluye a partir de la versión "esp_iot_sdk_v1.2.0". No es fiable trabajar con versiones antiguas, se recomienda ir actualizando el SDK.

En el sdk v1.5.1 vienen incluidas estas librerías:

Librería	Tamaño
libat.a	236 Kb
libcryto.a	80 Kb
libespnow.a	32 Kb
libjson.a	13 Kb
libmain.a	177 Kb
libmesh.a	158 Kb
libnet80211.a	254 Kb
libphy.a	147 Kb

libpp.a	223 Kb
libpwm.a	28 Kb
libsmartconfig.a	116 Kb
libssl.a	219 Kb
libupgrade.a	34 Kb
libwpa.a	125 Kb
libwpa2.a	267 Kb
libwps.a	228 Kb

Tabla 1 - Librerías "esp_iot_sdk_v1.5.1"

Para probar que funciona correctamente el protocolo WPS en el módulo ESP8266 se va utilizar el siguiente ejemplo “user_main.c”:

```
#include "osapi.h"
#include "user_interface.h"

#include "driver/key.h"

#define WPS_KEY_NUM        1

#define WPS_KEY_IO_MUX     PERIPHS_IO_MUX_MTCK_U
#define WPS_KEY_IO_NUM     13
#define WPS_KEY_IO_FUNC    FUNC_GPIO13

LOCAL struct keys_param keys;
LOCAL struct single_key_param *single_key;

LOCAL void ICACHE_FLASH_ATTR
user_wps_status_cb(int status)
{
      switch (status) {
            case WPS_CB_ST_SUCCESS:
                  wifi_wps_disable();
                  wifi_station_connect();
                  break;
            case WPS_CB_ST_FAILED:
            case WPS_CB_ST_TIMEOUT:
                  wifi_wps_start();
                  break;
      }
}

LOCAL void ICACHE_FLASH_ATTR
user_wps_key_short_press(void)
{
      wifi_wps_disable();
```

```
	wifi_wps_enable(WPS_TYPE_PBC);
	wifi_set_wps_cb(user_wps_status_cb);
	wifi_wps_start();
}

void ICACHE_FLASH_ATTR
user_rf_pre_init(void)
{
}

void ICACHE_FLASH_ATTR
user_init(void)
{
	single_key = key_init_single(WPS_KEY_IO_NUM, WPS_KEY_IO_MUX,
	WPS_KEY_IO_FUNC,NULL, user_wps_key_short_press);

	keys.key_num = WPS_KEY_NUM;
	keys.single_key = &single_key;

	key_init(&keys);

	wifi_set_opmode(STATION_MODE);
}
```

Este ejemplo es para probar WPS en el modo PBC. Lo que hace es ejecutar primero la función principal "void ICACHE_FLASH_ATTR user_init(void)", la cual pone al ESP8266 en modo estación y espera una pulsación corta en el botón.

Cuando el usuario pulsa el botón se salta a la función "LOCAL void ICACHE_FLASH_ATTR user_wps_key_short_press(void)", esta función hace las siguientes tareas en este orden, deshabilita el modo wps, lo vuelve a activar en modo PBC, llama a la función de callback de wps pasándole como parámetro otra función y arranca el protocolo wps para conectarse. Una vez realizadas estas tareas es cuando salta a la función "LOCAL void ICACHE_FLASH_ATTR user_wps_status_cb(int status)" en la que se comprueba si ha habido éxito en la conexión WPS, si ha fallado o si ha saltado el temporizador.

Este ejemplo se ha compilado en el entorno de desarrollo de eclipse configurado anteriormente. Para compilarlo se ha usado el siguiente "Makefile":

```
BUILD_BASE = build
FW_BASE = firmware

# Base directory for the compiler
XTENSA_TOOLS_ROOT ?= c:/Espressif/xtensa-lx106-elf/bin

# base directory of the ESP8266 SDK package, absolute
SDK_BASE	?= c:/Espressif/ESP8266_SDK
SDK_TOOLS	?= c:/Espressif/utils

# esptool path and port
ESPTOOL ?= $(SDK_TOOLS)/esptool.exe
ESPPORT ?= COM4
# Baud rate for programmer
BAUD ?= 115200
```

```
# SPI_SPEED = 40, 26, 20, 80
SPI_SPEED ?= 40
# SPI_MODE: qio, qout, dio, dout
SPI_MODE ?= qio
# SPI_SIZE_MAP
# 0 : 512 KB (256 KB + 256 KB)
# 1 : 256 KB
# 2 : 1024 KB (512 KB + 512 KB)
# 3 : 2048 KB (512 KB + 512 KB)
# 4 : 4096 KB (512 KB + 512 KB)
# 5 : 2048 KB (1024 KB + 1024 KB)
# 6 : 4096 KB (1024 KB + 1024 KB)
SPI_SIZE_MAP ?= 0

ifeq ($(SPI_SPEED), 26.7)
    freqdiv = 1
    flashimageoptions = -ff 26m
else
    ifeq ($(SPI_SPEED), 20)
        freqdiv = 2
        flashimageoptions = -ff 20m
    else
        ifeq ($(SPI_SPEED), 80)
            freqdiv = 15
            flashimageoptions = -ff 80m
        else
            freqdiv = 0
            flashimageoptions = -ff 40m
        endif
    endif
endif

ifeq ($(SPI_MODE), QOUT)
    mode = 1
    flashimageoptions += -fm qout
else
    ifeq ($(SPI_MODE), DIO)
           mode = 2
           flashimageoptions += -fm dio
    else
        ifeq ($(SPI_MODE), DOUT)
            mode = 3
            flashimageoptions += -fm dout
        else
            mode = 0
            flashimageoptions += -fm qio
        endif
    endif
endif

ifeq ($(SPI_SIZE_MAP), 1)
  size_map = 1
  flash = 256
  flashimageoptions += -fs 2m
else
  ifeq ($(SPI_SIZE_MAP), 2)
```

```
    size_map = 2
    flash = 1024
    flashimageoptions += -fs 8m
  else
    ifeq ($(SPI_SIZE_MAP), 3)
      size_map = 3
      flash = 2048
      flashimageoptions += -fs 16m
    else
      ifeq ($(SPI_SIZE_MAP), 4)
           size_map = 4
           flash = 4096
           flashimageoptions += -fs 32m
      else
        ifeq ($(SPI_SIZE_MAP), 5)
          size_map = 5
          flash = 2048
          flashimageoptions += -fs 16m
        else
          ifeq ($(SPI_SIZE_MAP), 6)
            size_map = 6
            flash = 4096
            flashimageoptions += -fs 32m
          else
            size_map = 0
            flash = 512
            flashimageoptions += -fs 4m
          endif
        endif
      endif
    endif
  endif
endif

# name for the target project
TARGET = app

# which modules (subdirectories) of the project to include in
compiling
MODULES    = driver user
EXTRA_INCDIR = include $(SDK_BASE)/../extra/include

# libraries used in this project, mainly provided by the SDK
LIBS = c gcc hal phy pp net80211 lwip wpa main smartconfig crypto wps

# compiler flags using during compilation of source files
CFLAGS = -Os -g -O2 -std=gnu90 -Wpointer-arith -Wundef -Werror -Wl,-EL
-fno-inline-functions -nostdlib -mlongcalls -mtext-section-literals -
mno-serialize-volatile -D__ets__ -DICACHE_FLASH

# linker flags used to generate the main object file
LDFLAGS = -nostdlib -Wl,--no-check-sections -u call_user_start -Wl,-
static

# linker script used for the above linkier step
LD_SCRIPT = eagle.app.v6.ld
```

```
# various paths from the SDK used in this project
SDK_LIBDIR = lib
SDK_LDDIR  = ld
SDK_INCDIR = include include/json

# select which tools to use as compiler, librarian and linker
CC    := $(XTENSA_TOOLS_ROOT)/xtensa-lx106-elf-gcc
AR    := $(XTENSA_TOOLS_ROOT)/xtensa-lx106-elf-ar
LD    := $(XTENSA_TOOLS_ROOT)/xtensa-lx106-elf-gcc
OBJCOPY := $(XTENSA_TOOLS_ROOT)/xtensa-lx106-elf-objcopy
OBJDUMP := $(XTENSA_TOOLS_ROOT)/xtensa-lx106-elf-objdump

# no user configurable options below here
SRC_DIR         := $(MODULES)
BUILD_DIR  := $(addprefix $(BUILD_BASE)/,$(MODULES))
SDK_LIBDIR := $(addprefix $(SDK_BASE)/,$(SDK_LIBDIR))
SDK_INCDIR := $(addprefix -I$(SDK_BASE)/,$(SDK_INCDIR))
SRC             := $(foreach sdir,$(SRC_DIR),$(wildcard $(sdir)/*.c))
OBJ             := $(patsubst %.c,$(BUILD_BASE)/%.o,$(SRC))
LIBS       := $(addprefix -l,$(LIBS))
APP_AR          := $(addprefix $(BUILD_BASE)/,$(TARGET)_app.a)
TARGET_OUT := $(addprefix $(BUILD_BASE)/,$(TARGET).out)

LD_SCRIPT  := $(addprefix -T$(SDK_BASE)/$(SDK_LDDIR)/,$(LD_SCRIPT))

INCDIR                := $(addprefix -I,$(SRC_DIR))
EXTRA_INCDIR    := $(addprefix -I,$(EXTRA_INCDIR))
MODULE_INCDIR   := $(addsuffix /include,$(INCDIR))

V ?= $(VERBOSE)
ifeq ("$(V)","1")
Q :=
vecho := @true
else
Q := @
vecho := @echo
endif

vpath %.c $(SRC_DIR)

define compile-objects
$1/%.o: %.c
	$(vecho) "CC $$<"
	$(Q) $(CC) $(INCDIR) $(MODULE_INCDIR) $(EXTRA_INCDIR)
$(SDK_INCDIR) $(CFLAGS)  -c $$< -o $$@
endef

.PHONY: all checkdirs clean flash flashinit flashonefile rebuild

all: checkdirs $(TARGET_OUT)

$(TARGET_OUT): $(APP_AR)
	$(vecho) "LD $@"
	$(Q) $(LD) -L$(SDK_LIBDIR) $(LD_SCRIPT) $(LDFLAGS) -Wl,--start-
group $(LIBS) $(APP_AR) -Wl,--end-group -o $@
	$(vecho) "-------------------------------------------------------
```

```
-----------------------"
	$(vecho) "Section info:"
	$(Q) $(OBJDUMP) -h -j .data -j .rodata -j .bss -j .text -j
.irom0.text $@
	$(vecho) "-------------------------------------------------------
-----------------------"
	$(Q) $(ESPTOOL) elf2image $(TARGET_OUT) -o$(FW_BASE)/
$(flashimageoptions)
	$(vecho) "-------------------------------------------------------
-----------------------"
	$(vecho) "Generate 0x00000.bin and 0x40000.bin successully in
folder $(FW_BASE)."
	$(vecho) "0x00000.bin-------->0x00000"
	$(vecho) "0x40000.bin-------->0x40000"
	$(vecho) "Done"

$(APP_AR): $(OBJ)
	$(vecho) "AR $@"
	$(Q) $(AR) cru $@ $^

checkdirs: $(BUILD_DIR) $(FW_BASE)

$(BUILD_DIR):
	$(Q) mkdir -p $@

$(FW_BASE):
	$(Q) mkdir -p $@

flashonefile: all
	$(SDK_TOOLS)/gen_flashbin.exe $(FW_BASE)/0x00000.bin
$(FW_BASE)/0x40000.bin 0x40000
	$(Q) mv eagle.app.flash.bin $(FW_BASE)/
	$(vecho) "Generate eagle.app.flash.bin successully in folder
$(FW_BASE)."
	$(vecho) "eagle.app.flash.bin-------->0x00000"
	$(ESPTOOL) -p $(ESPPORT) -b $(BAUD) write_flash
$(flashimageoptions) 0x00000 $(FW_BASE)/eagle.app.flash.bin

flash: all
	$(ESPTOOL) -p $(ESPPORT) -b $(BAUD) write_flash
$(flashimageoptions) 0x00000 $(FW_BASE)/0x00000.bin 0x40000
$(FW_BASE)/0x40000.bin

# ============================================================
# From http://bbs.espressif.com/viewtopic.php?f=10&t=305
# master-device-key.bin is only need if using espressive services
# master_device_key.bin 0x3e000 is not used , write blank
# See 2A-ESP8266__IOT_SDK_User_Manual__EN_v1.1.0.pdf
# http://bbs.espressif.com/download/file.php?id=532
#
# System parameter area is the last 16KB of flash
# 512KB flash - system parameter area starts from 0x7C000
#     download blank.bin to 0x7E000 as initialization.
# 1024KB flash - system parameter area starts from 0xFC000
#     download blank.bin to 0xFE000 as initialization.
# 2048KB flash - system parameter area starts from 0x1FC000
#     download blank.bin to 0x1FE000 as initialization.
```

```
# 4096KB flash - system parameter area starts from 0x3FC000
#     download blank.bin to 0x3FE000 as initialization.
# ===============================================================

# FLASH SIZE
flashinit:
	$(vecho) "Flash init data:"
	$(vecho) "Default config (Clear SDK settings):"
	$(vecho) "blank.bin-------->0x3e000"
	$(vecho) "blank.bin-------->0x3fc000"
	$(vecho) "esp_init_data_default.bin-------->0x3fc000"
	$(ESPTOOL) -p $(ESPPORT) write_flash $(flashimageoptions) \
		0x3e000 $(SDK_BASE)/bin/blank.bin \
		0x3fc000 $(SDK_BASE)/bin/esp_init_data_default.bin \
		0x3fe000 $(SDK_BASE)/bin/blank.bin

rebuild: clean all

clean:
	$(Q) rm -f $(APP_AR)
	$(Q) rm -f $(TARGET_OUT)
	$(Q) rm -rf $(BUILD_DIR)
	$(Q) rm -rf $(BUILD_BASE)
	$(Q) rm -rf $(FW_BASE)

$(foreach bdir,$(BUILD_DIR),$(eval $(call compile-objects,$(bdir))))
```

Con este makefile podemos tanto compilar el código con la opción “all”, borrar los ficheros con la opción “clean” como incluso flashear el código con la opción “flash”. Únicamente hay que seleccionar el puerto correcto y poner los baudios adecuados.

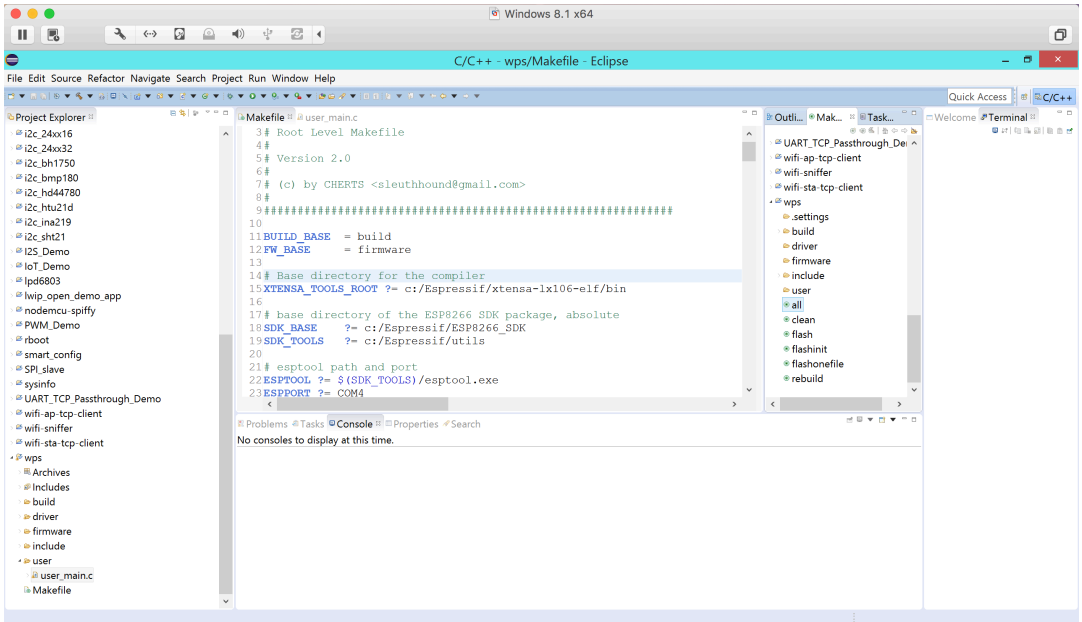


Figura 15 - Ejemplo WPS en eclipse

Una vez flasheado el binario generado en el ESP8266, para probarlo basta con pulsar el botón de wps del router y después pulsar el botón que se tiene conectado al GPIO13, ya que este es el GPIO que se ha elegido

en el código “user_macin.c”. Tras unos segundos el dispositivo ESP8266 se conectará al router y tendremos acceso a internet.

4.2 Como crear el nuevo módulo WPS

El firmware de NodeMCU contiene un bastantes módulos y aplicaciones, todos ellos incluidos en la carpeta “app”. Dentro de esta carpeta en “nodemcu-firmware/app/modules” es donde encontramos los módulos contenidos en este software. Es aquí donde se añadirá el nuevo módulo WPS, para ello se crea el fichero “wps.c” dentro de “nodemcu-firmware/app/modules”.

El código de este fichero es:

```
/ Modulo para el uso del protocolo WPS

#include "module.h"
#include "lauxlib.h"
#include "platform.h"

#include "c_string.h"
#include "c_stdlib.h"
#include "ctype.h"

#include "c_types.h"
#include "user_interface.h"

// Funcion de saludo de wps
static int ICACHE_FLASH_ATTR wps_saludo(lua_State* L)
{
  lua_pushstring(L, "Hola desde el módulo WPS.");
  return 1;
}

// Module function map
const LUA_REG_TYPE wps_map[] =  {
  { LSTRKEY( "saludo" ),         LFUNCVAL( wps_saludo ) },
{ LNILKEY, LNILVAL }
};

int luaopen_wps( lua_State *L )
{
 return 0;
}


NODEMCU_MODULE(WPS, "wps", wps_map, luaopen_wps);
```

La primera parte del código es C, se incluyen las librerías que se van a utilizar, se definen las funciones y su contenido.

La segunda parte es la conexión entre el código C y Lua, debemos hacer que Lua conozca de alguna forma que métodos son accesibles para ella. Esto se hace a través de “const LUA_REG_TYPE wps_map[]”. Se está exponiendo una función en este caso “wps_saludo”. Ésta será registrada cuando se importe el módulo "NODEMCU_MODULE(WPS, "wps", wps_map, luaopen_wps).

4.2.1 Integrar el módulo en NodeMCU

Cuando se compila, nuestro módulo será parte de la librería “libmodules.a”. Pero eso no significa que vaya a ser incluido automáticamente en el firmware generado. Eso depende del fichero “app/include/user_modules.h” cabecera. Es en él donde se deciden que módulos son incluidos y cuáles no. Para que nuestro módulo sea incluido basta con añadir la siguiente declaración:

```
#define LUA_USE_MODULES_WPS // Modulo WPS añadido
```

El fichero “app/include/user_modules.h” resultante es (hay módulos comentados porque no se usan y así se reduce el tamaño de la memoria RAM, ya que está bastante limitada):

```
#ifndef __USER_MODULES_H__
#define __USER_MODULES_H__

#define LUA_USE_BUILTIN_STRING        // for string.xxx()
#define LUA_USE_BUILTIN_TABLE         // for table.xxx()
#define LUA_USE_BUILTIN_COROUTINE     // for coroutine.xxx()
#define LUA_USE_BUILTIN_MATH          // for math.xxx(), partially work
// #define LUA_USE_BUILTIN_IO              // for io.xxx(), partially
work

// #define LUA_USE_BUILTIN_OS              // for os.xxx(), not work
// #define LUA_USE_BUILTIN_DEBUG
#define LUA_USE_BUILTIN_DEBUG_MINIMAL // for debug.getregistry() and
debug.traceback()

#ifndef LUA_CROSS_COMPILER

// The default configuration is designed to run on all ESP modules
including the 512 KB modules like ESP-01 and only
// includes general purpose interface modules which require at most
two GPIO pins.
// See https://github.com/nodemcu/nodemcu-firmware/pull/1127 for
discussions.
// New modules should be disabled by default and added in alphabetical
order.
#define LUA_USE_MODULES_ADC
//#define LUA_USE_MODULES_AM2320
//#define LUA_USE_MODULES_APA102
#define LUA_USE_MODULES_BIT
//#define LUA_USE_MODULES_BMP085
//#define LUA_USE_MODULES_BME280
//#define LUA_USE_MODULES_CJSON
//#define LUA_USE_MODULES_COAP
//#define LUA_USE_MODULES_CRYPTO
#define LUA_USE_MODULES_DHT
//#define LUA_USE_MODULES_ENCODER
#define LUA_USE_MODULES_ENDUSER_SETUP // USE_DNS in dhcpserver.h needs
to be enabled for this module to work.
#define LUA_USE_MODULES_FILE
#define LUA_USE_MODULES_GPIO
//#define LUA_USE_MODULES_HTTP
//#define LUA_USE_MODULES_HX711
#define LUA_USE_MODULES_I2C
```

```
//#define LUA_USE_MODULES_MDNS
#define LUA_USE_MODULES_MQTT
#define LUA_USE_MODULES_NET
#define LUA_USE_MODULES_NODE
#define LUA_USE_MODULES_OW
//#define LUA_USE_MODULES_PERF
//#define LUA_USE_MODULES_PWM
//#define LUA_USE_MODULES_RC
//#define LUA_USE_MODULES_ROTARY
//#define LUA_USE_MODULES_RTCFIFO
//#define LUA_USE_MODULES_RTCMEM
//#define LUA_USE_MODULES_RTCTIME
//#define LUA_USE_MODULES_SIGMA_DELTA
//#define LUA_USE_MODULES_SNTP
#define LUA_USE_MODULES_SPI
//#define LUA_USE_MODULES_STRUCT
#define LUA_USE_MODULES_TMR
//#define LUA_USE_MODULES_TSL2561
//#define LUA_USE_MODULES_U8G
#define LUA_USE_MODULES_UART
//#define LUA_USE_MODULES_UCG
#define LUA_USE_MODULES_WIFI
//#define LUA_USE_MODULES_WS2801
//#define LUA_USE_MODULES_WS2812
#define LUA_USE_MODULES_WPS // Modulo WPS añadido


#endif  /* LUA_CROSS_COMPILER */

#endif  /*__USER_MODULES_H__  */
```

Una vez realizados estos cambios, ya tenemos un módulo nuevo llamado “WPS”, con una única función hasta el momento “wps_saludo”. Este módulo ya se encuentra incluido en el fichero “app/include/user_modules.h” y por tanto en el firmware de NodeMCU. Ahora se van a probar los cambios realizados hasta el momento.

Para comprobar que se ha creado bien el módulo y la función, y se ha integrado en el firmware de NodeMCU correctamente, se compila el código (usando el docker). Se flashea el binario generado en el ESP8266, se conecta el ESP al ordenador y se llama a la función creada “wps.saludo”:

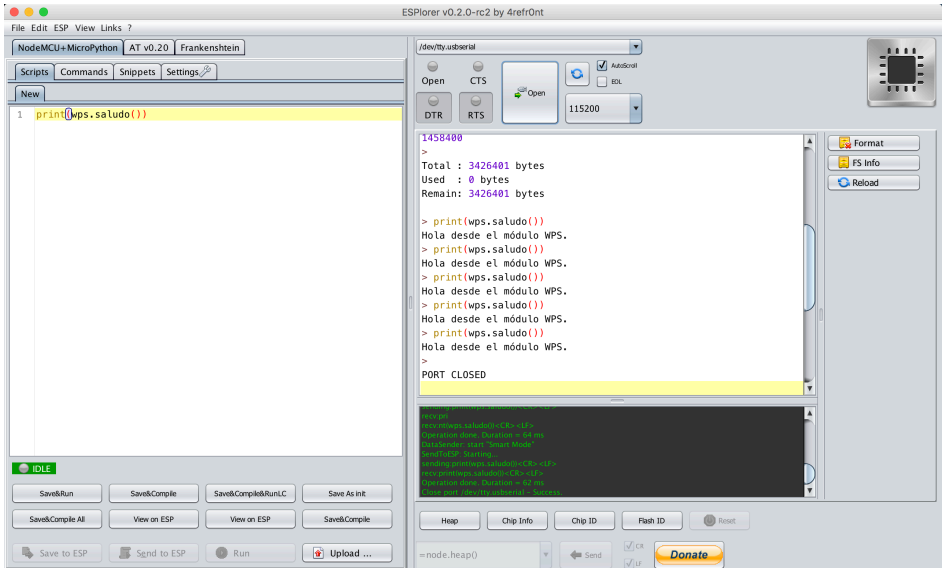


Figura 16 - Probando el módulo "wps"

4.3 Construyendo el módulo WPS

Ya se ha creado el módulo wps y se ha integrado dentro del NodeMCU, solo se dispone de una función básica que sea creado para probar que el módulo funciona correctamente, ahora es necesario incluir en este código el resto de funciones. Para ello es necesario incluir la librería "libwps.a" para hacer uso de sus funciones.

Para ello se añadirá la librería al makefile "nodemcu-firmware/app/Makefile":

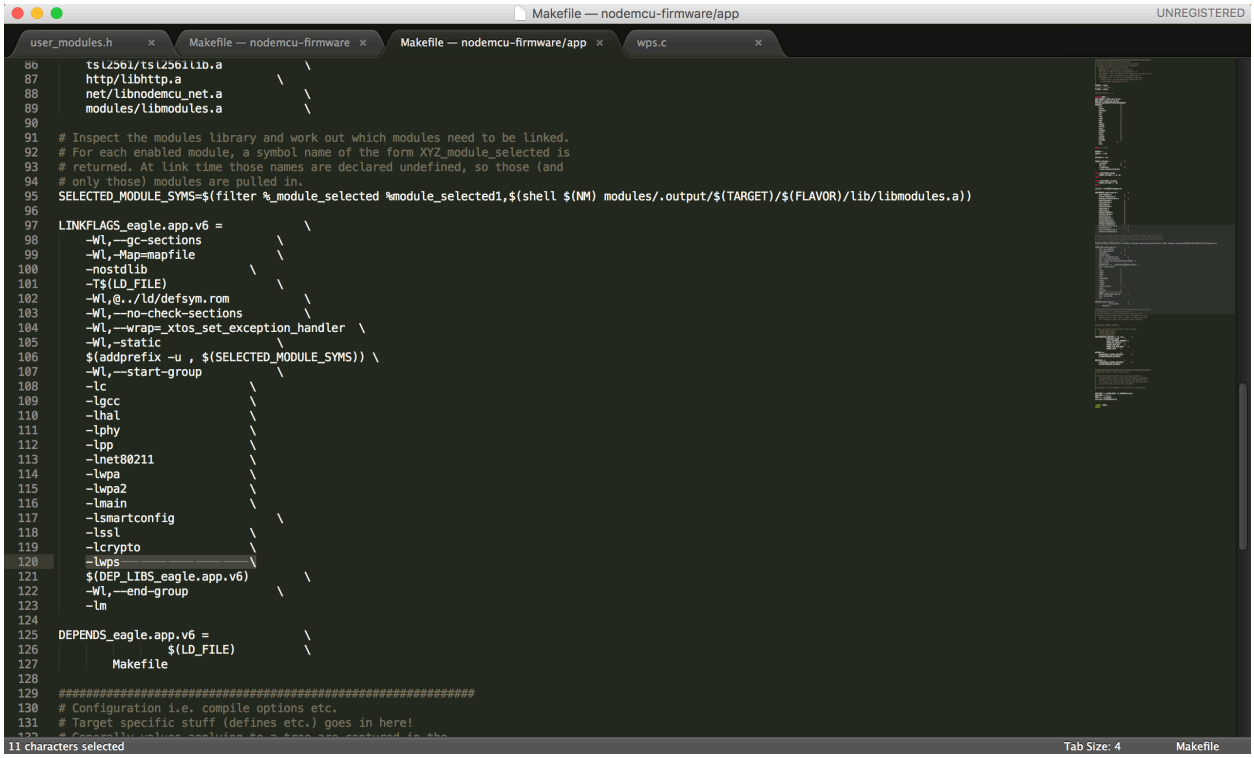


Figura 17 - Añadir "libwps.a"

Se añade la línea 120 a este fichero. Ahora cuando compilemos el código la librería estará añadida.

Ya se puede hacer uso de las funciones wps incluidas en la librería, las que se usarán son: [17]

60. wifi_wps_disable

Function:

Disable Wi-Fi WPS function and release resource it taken

Prototype:

```
bool wifi_wps_disable(void)
```

Parameter:

none

Return:

true: succeed
false: fail

Figura 18 - wifi_wps_disable

59. wifi_wps_enable

Function:

Enable Wi-Fi WPS function

Note:

WPS can only be used when ESP8266 station is enabled.

Structure:

```
typedef enum wps_type {
    WPS_TYPE_DISABLE=0,
    WPS_TYPE_PBC,
    WPS_TYPE_PIN,
    WPS_TYPE_DISPLAY,
    WPS_TYPE_MAX,
}WPS_TYPE_t;
```

Prototype:

```
bool wifi_wps_enable(WPS_TYPE_t wps_type)
```

Parameter:

`WPS_TYPE_t wps_type` : WPS type, so far only `WPS_TYPE_PBC` is supported

Return:

true: succeed
false: fail

Figura 19 - wifi_wps_enable

61. wifi_wps_start

Function:

WPS starts to work

Note:

WPS can only be used when ESP8266 station is enabled.

Prototype:

```
bool wifi_wps_start(void)
```

Parameter:

none

Return:

true: means that WPS starts to work successfully, does not mean WPS succeed.

false: fail

Figura 20 - wifi_wps_start

62. wifi_set_wps_cb

Function:

Set WPS callback, parameter of the callback is the status of WPS.

Callback and parameter structure:

```
typedef void (*wps_st_cb_t)(int status);

enum wps_cb_status {
    WPS_CB_ST_SUCCESS = 0,
    WPS_CB_ST_FAILED,
    WPS_CB_ST_TIMEOUT,
    WPS_CB_ST_WEP,   // WPS failed because that WEP is not supported
    WPS_CB_ST_SCAN_ERR, // can not find the target WPS AP
};
```

Note:

- If parameter `status == WPS_CB_ST_SUCCESS` in WPS callback, it means WPS got AP's information, user can call `wifi_wps_disable` to disable WPS and release resource, then call `wifi_station_connect` to connect to target AP.
- Otherwise, it means that WPS fail, user can create a timer to retry WPS by `wifi_wps_start` after a while, or call `wifi_wps_disable` to disable WPS and release resource.

Prototype:

```
bool wifi_set_wps_cb(wps_st_cb_t cb)
```

Parameter:

`wps_st_cb_t cb` : callback

Return:

true: succeed
false: fail

Figura 21 - wifi_set_wps_cb

Ahora se crean 4 funciones en el módulo wps, cada una de ellas llamará a una de éstas, será necesario crear una función más, ya que la función "wifi_set_wps_cb" recibe como parámetro una función. Para que Lua las identifique y sepa como llamarlas es necesario añadir las 5 funciones al "const LUA_REG_TYPE wps_map[]".

El fichero "nodemcu-firmware/app/modules/wps.c" quedará de la siguiente:

```
// Modulo para el uso del protocolo WPS

#include "module.h"
#include "lauxlib.h"
#include "platform.h"

#include "c_string.h"
#include "c_stdlib.h"
#include "ctype.h"
```

```
#include "c_types.h"
#include "user_interface.h"

// Funcion de saludo de wps
static int ICACHE_FLASH_ATTR wps_saludo(lua_State* L)
{
  lua_pushstring(L, "Hola desde el módulo WPS.");
  return 1;
}

// Funcion para deshabilitar el wps
static int ICACHE_FLASH_ATTR wps_disable(lua_State* L)
{
  unsigned int estado;

  estado=wifi_wps_disable();
  return estado;
}

// Funcion para habilitar el wps
static int ICACHE_FLASH_ATTR wps_enable(lua_State* L)
{
  unsigned int estado;

  estado=wifi_wps_enable(WPS_TYPE_PBC);
  return estado;
}

// Funcion de callback del wps
LOCAL void ICACHE_FLASH_ATTR user_wps_status_cb(int status)
{
  switch (status) {
    case WPS_CB_ST_SUCCESS:
      wifi_wps_disable();
      wifi_station_connect();
      break;
    case WPS_CB_ST_FAILED:
    case WPS_CB_ST_TIMEOUT:
      wifi_wps_start();
      break;
  }
}

// Funcion de llamada
static int ICACHE_FLASH_ATTR wps_set(lua_State* L)
{
  unsigned int estado;

  estado=wifi_set_wps_cb(user_wps_status_cb);
  return estado;
}

// Función para arrancar el wps
static int ICACHE_FLASH_ATTR wps_start(lua_State* L)
{
  unsigned int estado;
```

```
  estado=wifi_wps_start();
  return estado;
}

// Module function map
const LUA_REG_TYPE wps_map[] =  {
  { LSTRKEY( "saludo" ),        LFUNCVAL( wps_saludo ) },
  { LSTRKEY( "disable" ),        LFUNCVAL( wps_disable ) },
  { LSTRKEY( "enable" ),        LFUNCVAL( wps_enable ) },
  { LSTRKEY( "set" ),        LFUNCVAL( wps_set ) },
  { LSTRKEY( "start" ),        LFUNCVAL( wps_start ) },
  { LNILKEY, LNILVAL }
};

int luaopen_wps( lua_State *L )
{
 return 0;
}

NODEMCU_MODULE(WPS, "wps", wps_map, luaopen_wps);
```

Se ha implementado el modo wps PBC, que hasta ahora es el único soportado por “wifi_wps_enable”. Cuando se compile el código dispondremos de los binarios del nodemcu_firmware pero ahora con estás 5 funciones nuevas:

```
wps.saludo()
wps.disable()
wps.enable()
wps.set()
wps.start()
```

Para conectar nuestro módulo al router por wps hay que activar en el router el modo wps, esto se hace bien activándolo desde la página de configuración del router o pulsando el botón wps si lo tiene, esto permitirá que durante unos minutos puedan conectarse los dispositivos a él. Durante este intervalo es necesario ejecutar las 4 funciones en el orden puesto anteriormente y el ESP8266 se conectará al router.

5 PRUEBAS

5.1 Entorno de pruebas

Se dispondrá de un ESP8266 con el nuevo firmware de NodeMCU modificado que incluye el protocolo WPS y para probar su funcionamiento se usará un router que permita este protocolo (actualmente casi todos los router cuentan con WPS), en concreto el modelo usado es el "TP LINK TL-WR841ND".

Figura 22 - TP LINK TL-WR841ND

Ahora se realizarán distintas pruebas a la hora conectar el ESP8266 al router por WPS, se presentarán 3 escenarios y se verá como reacciona el ESP8266 ante éstas situaciones.

5.2 Pruebas

Se intentará conectar el ESP8266 al router en tres escenarios distintos, se presentan a continuación:

5.2.1 Escenario 1: Situación normal

Este es el funcionamiento normal si los dos dispositivos funcionan correctamente. Para conectar el ESP al router hay que activar la opción de WPS en el router, esto se hace en este router mediante la página de configuración de éste. Se accede al router y se va a la pestaña de WPS:

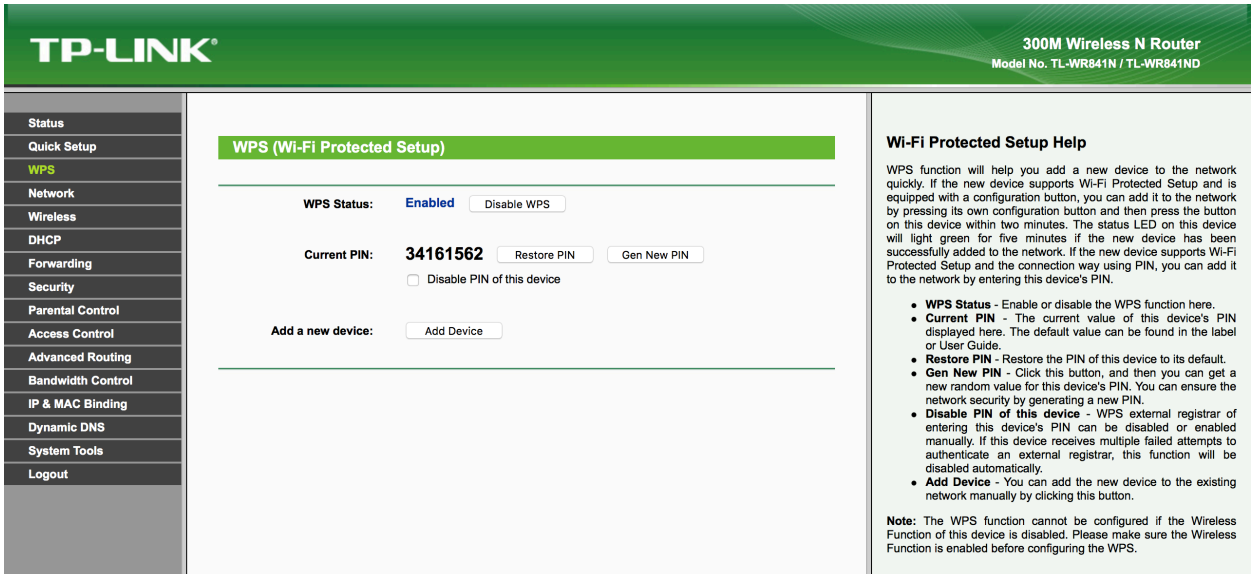


Figura 23 - Activando WPS en el router

Se activa el WPS (“enabled”) y se accede a la pestaña “Add Device”, donde aparece la siguiente ventana:

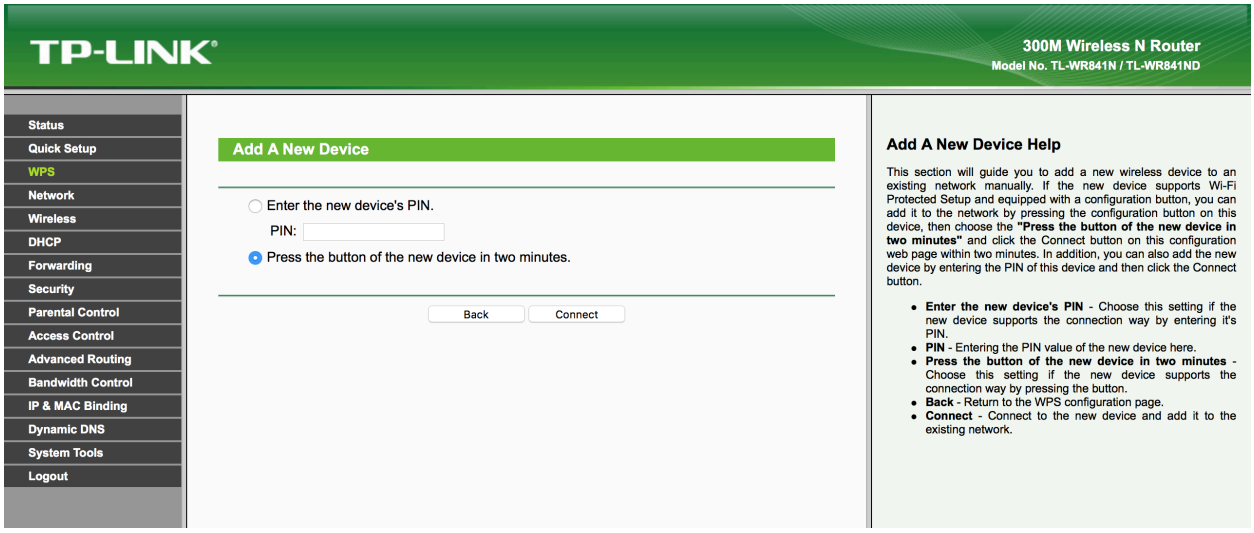


Figura 24 - Añadir nuevo dispositivo

Se elige el modo de conexión PBC y se le da a connect:

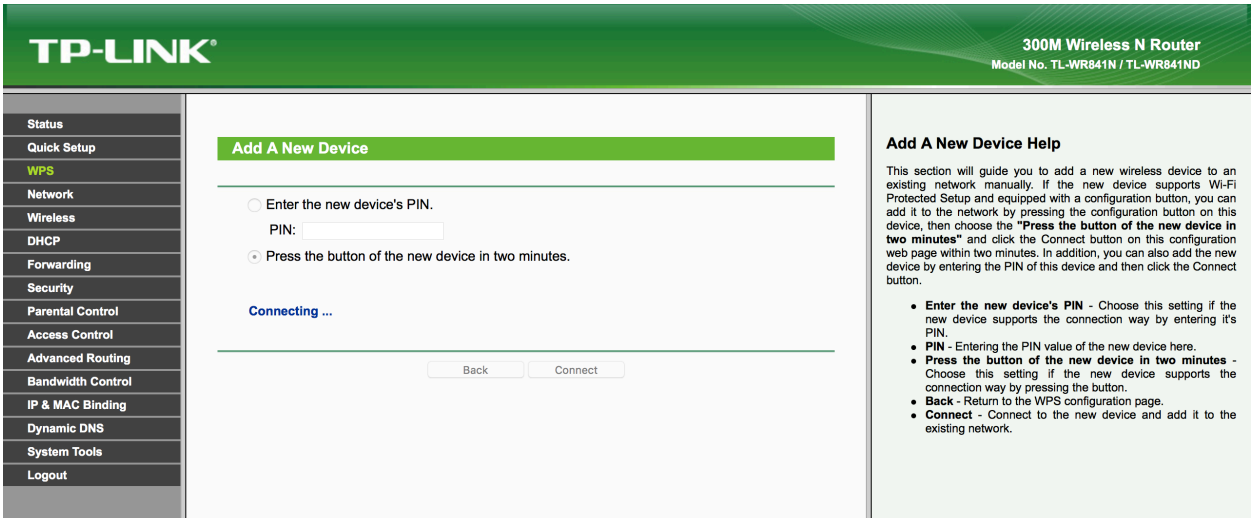


Figura 25 - Conectando

Ahora el router permite que durante un intervalo de dos minutos los dispositivos se conecten a el por WPS. Se conecta el ESP8266 al ordenador con "ESPlorer", se prueban algunos comandos para ver que responde bien el ESP, probamos también el de la función de saludo desde wps y después se llama a las 4 funciones que se han creado:

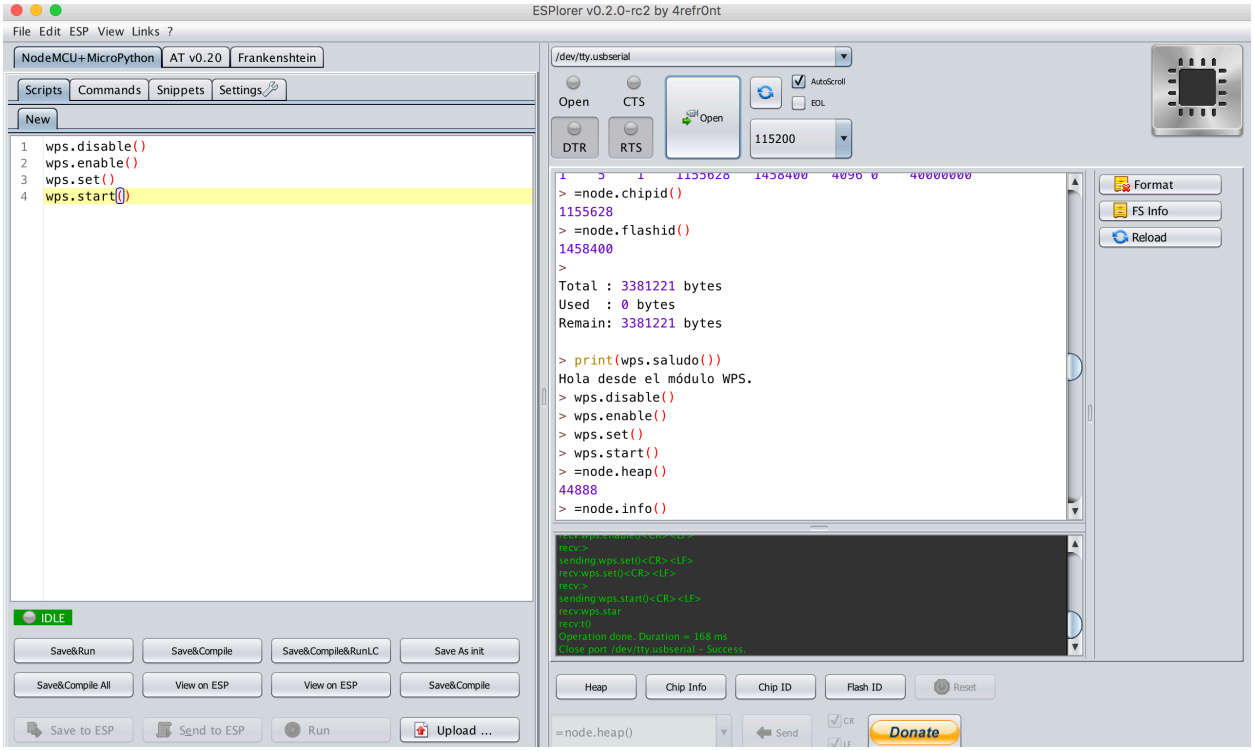


Figura 26 - Conectando el ESP8266 al router

Hay que esperar unos segundos a que se ejecuten las instrucciones y se ve la reacción en la página del router, que aparece lo siguiente:

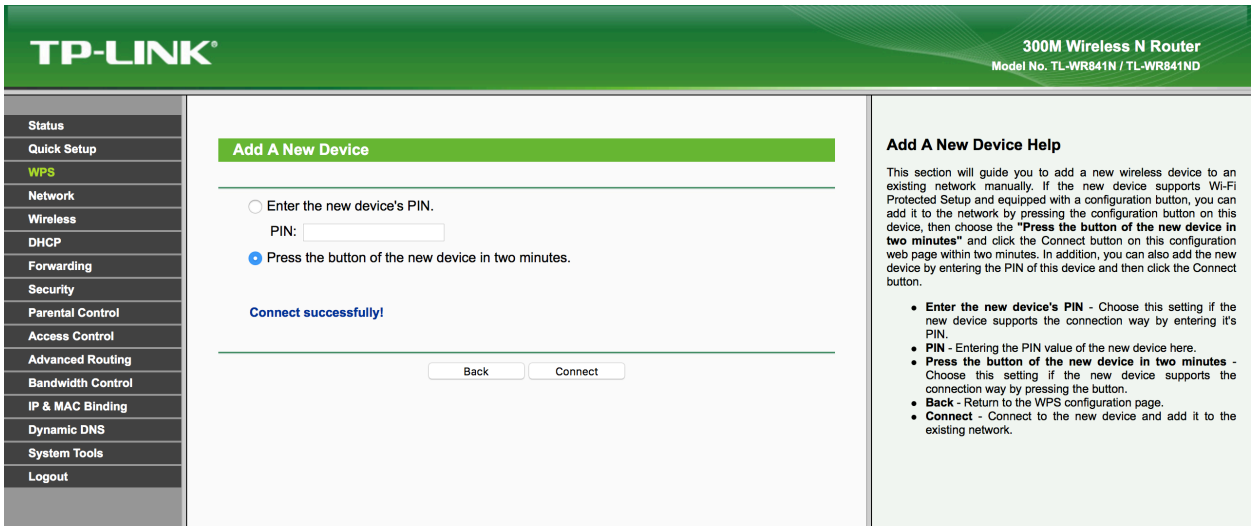


Figura 27 - Conectado con éxito

Se comprueba que hay un dispositivo ahora conectado y que su mac es la del ESP8266:

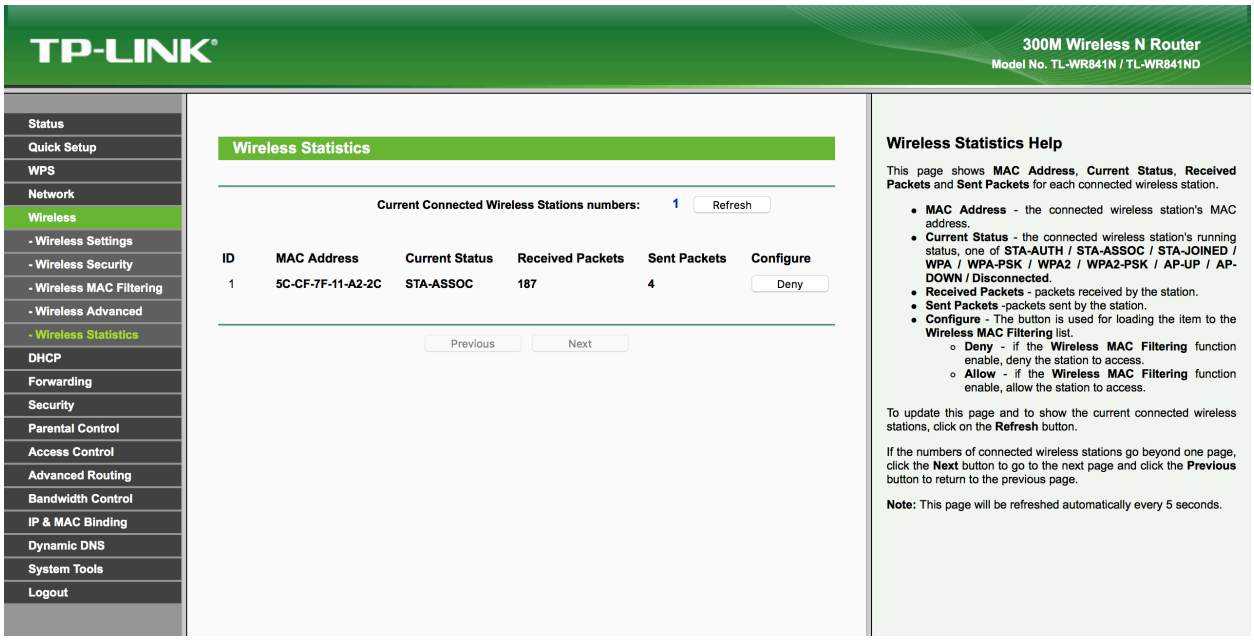


Figura 28 - ESP8266 conectado al router

Desde el ESP8266 también se puede comprobar que ha tenido éxito la conexión fijándose en lo que devuelve la función wps.set(), si ha tenido éxito y se ha conectado al router devolverá ”true” y en caso contrario devolverá “false”.

5.2.2 Escenario 2: Se apaga el ESP8266 durante la conexión

En este caso se repiten todos los pasos anteriores para activar el WPS en el router, una vez que está activo y el ESP8266 está conectado al ordenador, se tiran igual que antes las 4 funciones creadas desde el ESP, dentro del intervalo de los 2 minutos de conexión del router, pero ahora mientras se intenta establecer esa conexión se desconecta el ESP8266, la reacción se puede comprobar en la página del router que aparece:

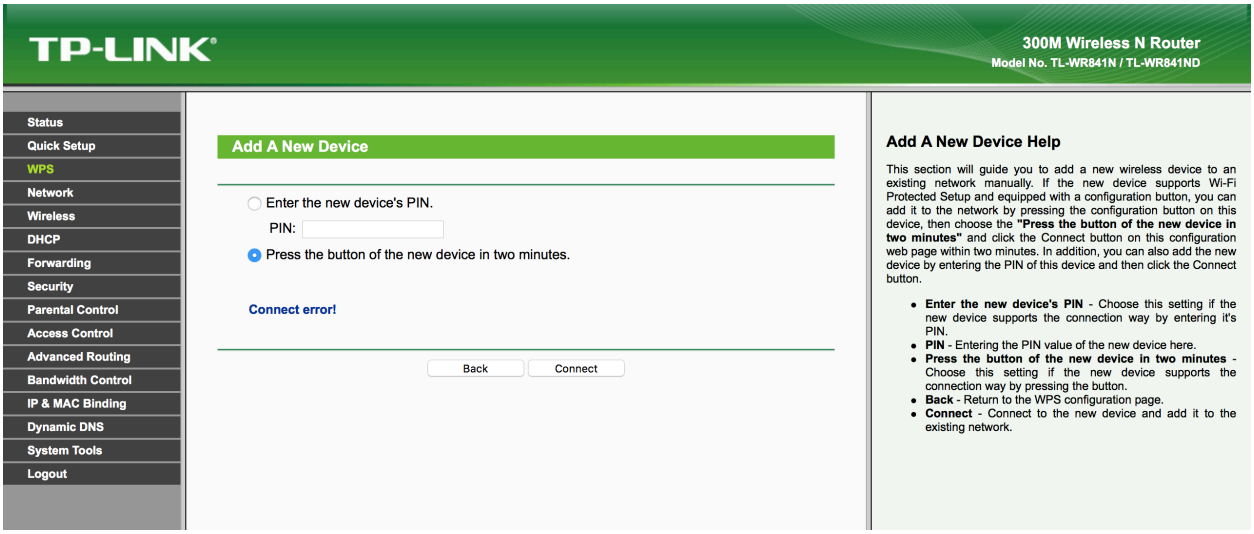


Figura 29 - Error en la conexión

Al volver a conectar el ESP8266 al ordenador, si se quiere conectar al router habrá que repetir los pasos del escenario 1.

5.2.3 Escenario 3: Se apaga el router durante la conexión

En este escenario se actúa igual que en los anteriores, se comienza activando el WPS desde la página del router como en el escenario 1, se conecta el ESP al ordenador y se tiran las 4 funciones dentro del intervalo de los dos minutos. Pero mientras se intenta realizar la conexión se apaga el router.

En este caso hay que fijarse en la función "wps.set()", el código de esta función es el siguiente:

```
// Funcion de llamada
static int ICACHE_FLASH_ATTR wps_set(lua_State* L)
{
  unsigned int estado;

  estado=wifi_set_wps_cb(user_wps_status_cb);
  return estado;
}
```

Esta función llama a la función de callback de wps y le pasa como parámetro la función "user_wps_status_cb":

```
// Funcion de callback del wps
LOCAL void ICACHE_FLASH_ATTR user_wps_status_cb(int status)
{
```

```
  switch (status) {
    case WPS_CB_ST_SUCCESS:
      wifi_wps_disable();
      wifi_station_connect();
      break;
    case WPS_CB_ST_FAILED:
    case WPS_CB_ST_TIMEOUT:
      wifi_wps_start();
      break;
  }
}
```

Es en esta función donde se decide que hacer, si el parámetro “status” vale “WPS_CB_ST_SUCESS” es por que todo ha ido bien y ha conseguido la información necesaria del punto de acceso y lo que se hace es desactivar el WPS para liberar recursos y llamar a “wifi_station_connect” para conectarse al router. Este es el caso del escenario 1.

O de lo contrario se ha producido un error sea por el motivo que sea (que es la situación de este escenario) y lo que se puede hacer es elegir una de las dos opciones que se dan en las notas de la función “wifi_set_wps_cb”:

1. Se intenta volver a conectar llamando a wifi_wps_start().
2. Se llama a la función wifi_wps_disable para desactivar el WPS y liberar recursos.

Si ha tenido éxito y se ha conectado al router, el parámetro “status” vale “WPS_CB_ST_SUCESS” y la función “wps.set()” devuelve “true” (escenario 1).

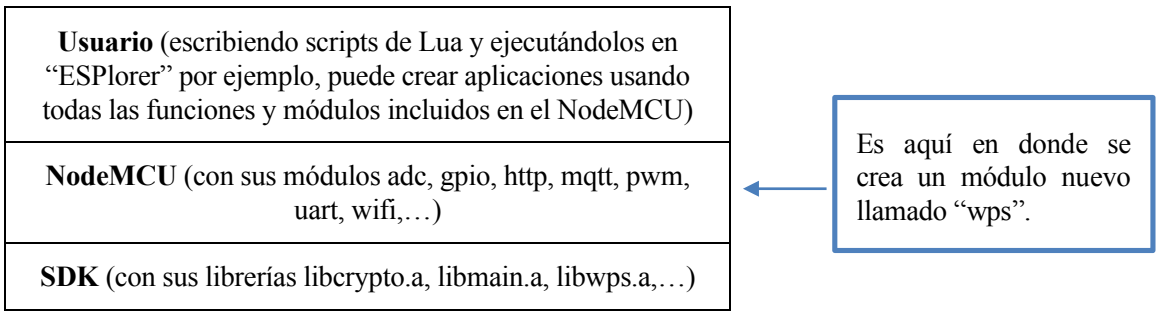
Si por el contrario ha fallado el valor de “status” será cualquier otro y la función “wps.set()” devuelve “false” (escenario actual 3). Se ha decido intentar volver a conectarse, de manera que tras un fallo el ESP8266 intentará conectarse hasta que se llame a la función “wifi.disable()” o se conecte con éxito.

6 CONCLUSIONES Y DESARROLLOS FUTUROS

6.1 Conclusiones

Por último para concluir este trabajo hacer un pequeño recordatorio de lo que se ha llevado a cabo. El trabajo consiste en añadir la funcionalidad de WPS al firmware de NodeMCU, para llevar a cabo esta tarea se han tenido que dar los siguientes pasos:

- Familiarizarse con el system on chip ESP8266 (sus modos de funcionamiento, configuración de los pines, modo flash, …).
- Este SOC cuenta con un SDK proporcionado por la compañía Espressif para programar directamente el dispositivo. En concreto se ha usado la versión "esp_iot_sdk_v1.5.1", se ha estudiado este firmware y se han probado varios ejemplos con él, entre ellos el que más nos interesa, el funcionamiento de WPS. Es posible hacer uso de WPS gracias a la librería "libwps.a" incluida en este SDK.
- A continuación se ha investigado en el firmware de NodeMCU, otro software para el ESP8266 que permite desarrollar aplicaciones en un lenguaje de alto nivel como es Lua. Este software está basado en el SDK "esp_iot_sdk_v1.5.1", y aunque ofrece casi todas las herramientas que contiene el SDK, en concreto la funcionalidad WPS no está incluida en el NodeMCU.
- Comienza aquí el objetivo del trabajo, incluir esa funcionalidad WPS en el NodeMCU. La estructura desde la que se parte es la siguiente:



Esta sería la estructura que se tiene, se parte del SDK y lo que se pretende es crear un módulo nuevo en el NodeMCU que haga uso de la librería "libwps.a". Para hacer esto se usa el lenguaje de programación Lua, que permite a las funciones creadas en el módulo "wps" llamar a las funciones incluidas en las librerías del SDK. Esta es la tarea de Lua, organizar y conectar el código del NodeMCU con el SDK, para permitir al NodeMCU hacer uso de las funciones incluidas en las librerías. Una vez realizado el trabajo lo que se obtiene es un módulo nuevo con la funcionalidad WPS dentro del NodeMCU.

Finalmente recordar que el abanico de posibilidades y aplicaciones que ofrece el ESP8266 es muy amplío, como se indica en su datasheet se puede incluir en electrodomésticos, automatización del hogar, enchufes y luces inteligentes, control industrial inalámbrico, monitores del bebé, cámaras IP, redes de sensores, electrónica portable, dispositivos de localización basados en Wi-Fi, identificación de etiquetas para temas de seguridad, sistemas de posicionamiento con balizas Wi-Fi,…

No hay que olvidar las principales características de este dispositivo como son su reducido coste, tamaño y la posibilidad de mantener este dispositivo conectado a Internet, hacen del ESP8266 un excelente SOC para el mundo de IoT y de los sistemas embebidos, ofreciendo un sin fin de aplicaciones y posibilidades en este nuevo mercado que no parará de crecer en los próximos años.

6.2 Desarrollos futuros

De cara a desarrollos futuros, una mejora importante es que en las próximas actualizaciones del SDK se prevé que soporte también el modo de conexión WPS por PIN, actualmente sólo es soportado el PBC y es el que ha sido implementado. Como mejora futura se propone actualizar el módulo WPS creado, para que también sea capaz de soportar la conexión por PIN.

Actualmente existen numerosos proyectos que se están llevando a cabo con el ESP8266. [18] Como posible trabajo futuro, ahora que ya se conoce mejor este dispositivo, se podría intentar llevar a cabo alguno de esos proyectos para poner el dispositivo en una aplicación concreta. Entre ellos se pueden destacar:

- Sistema de posicionamiento SubPos: es un sistema de posicionamiento en interiores que se puede utilizar en diversos entornos, tales como líneas de metro, centros comerciales, aparcamientos, galerías de arte o centros de conferencias, esencialmente en cualquier lugar del GPS no penetra. SubPos es un sistema de posicionamiento completo que no requiere la concesión de licencias caras, hardware especializado o de perfiles de área laborioso y no depende de la conectividad de datos El estándar define un método para la colocación subterránea en diferentes ambientes mediante la explotación de todas las capacidades de Wi-Fi. Plug and play SubPos nodos o puntos de acceso Wi-Fi existentes se utilizan para transmitir información codificada en un marco estándar de baliza Wi-Fi que luego se utiliza para el cálculo de la posición pasiva por un dispositivo de su elección. El ESP8266 es usado para crear el nodo SubPos. [19] [20]
- También existen aplicaciones en entornos domésticos Smart Home, orientadas a Smart Light y Smart Plug (para el control de las luces del hogar mediante un mando a distancia, el teléfono móvil mediante una app podría actuar de mando a distancia). [21]

REFERENCIAS

[1] Espressif Inc., “Overview ESP8266” [En línea]. Disponible en: https://espressif.com/en/products/hardware/esp8266ex/overview.

[2] Espressif Inc., “Datasheet del ESP8266” [En línea]. Disponible en: https://espressif.com/sites/default/files/documentation/0a-esp8266ex_datasheet_en.pdf.

[3] Mxchip, “ Overview EMW3165” [En línea]. Disponible en: http://en.mxchip.com.

[4] ST Microelectronics, “Datasheet del ARM Cortex M4 de 32 bits STM32F411CE” [En línea]. Disponible en: http://www.st.com/web/catalog/mmc/FM141/SC1169/SS1577/LN1877/PF260148.

[5] Mxchip, “Datasheet del EMW3165”. Disponible en: http://en.mxchip.com/product/wifi_product/38.

[6] “Conjunto de comandos AT”. [En línea]. Disponible en: http://www.lammertbies.nl/comm/info/hayes-at-commands.html.

[7] Arduino, “Guía” [En línea]. Disponible en: https://www.arduino.cc/en/Guide/HomePage.

[8] NodeMCU, “Firmware” [En línea]. Disponible en: https://github.com/nodemcu/nodemcu-firmware.

[9] NodeMCU, “Página oficial” [En línea]. Disponible en: http://nodemcu.com/index_en.html.

[10] “The programming language Lua” [En línea]. Disponible en: https://www.lua.org/about.html.

[11] Roberto Ierusalimschy, Luiz Henrique de Figueiredo y Waldemar Celes. “Manual de Referencia de Lua 5.1” [En línea]. Disponible en: https://www.lua.org/manual/5.1/es/manual.html.

[12] Wi-Fi Protected Setup [En línea]. Disponible en: https://es.wikipedia.org/wiki/Wi-Fi_Protected_Setup.

[13] Object Technology International, Inc. ·Eclispe Platform Thecnical Overview” [En línea]. Disponible en: http://web.archive.org/web/20130402233256/http://www.eclipse.org/whitepapers/eclipse-overview.pdf.

[14] Neil Kolba, “Kolban’s Book on ESP8266”, Enero 2016.

[15] Marcel Stör, “Docker for Mac” [En línea]. Disponible en: https://docs.docker.com/engine/installation/mac/#/docker-toolbox.

[16] Marcel Stör, “NodeMCU Build” [En línea]. Disponible en: https://hub.docker.com/r/marcelstoer/nodemcu-build/.

[17] Espressif System IOT Team, “ESP8266 SDK Api Guide”, Enero 2016.

[18] Proyectos con el ESP8266 [En línea]. Disponible en: https://hackaday.io/projects/tag/ESP8266/sort/views.

[19] “Sistema de posicionamiento SubPos” [En línea]. Disponible: https://hackaday.io/project/4872-subpos-positioning-system.

[20] Página oficial de SubPos [En línea]. Disponible en: http://subpos.org.

[21] Espressif, “Smart Home” [En línea]. Disponible en: http://www.espressif.com/en/products/industries/smart-home.

ANEXO A: CÓDIGO

nodemcu-firmware/app/Makefile:

```
#############################################################
# Required variables for each makefile
# Discard this section from all parent makefiles
# Expected variables (with automatic defaults):
#   CSRCS (all "C" files in the dir)
#   SUBDIRS (all subdirs with a Makefile)
#   GEN_LIBS - list of libs to be generated ()
#   GEN_IMAGES - list of object file images to be generated ()
#   GEN_BINS - list of binaries to be generated ()
#   COMPONENTS_xxx - a list of libs/objs in the form
#     subdir/lib to be extracted and rolled up into
#     a generated lib/image xxx.a ()
#
TARGET = eagle
#FLAVOR = release
FLAVOR = debug

#EXTRA_CCFLAGS += -u

ifndef PDIR # {
GEN_IMAGES= eagle.app.v6.out
GEN_BINS= eagle.app.v6.bin
SPECIAL_MKTARGETS=$(APP_MKTARGETS)
SUBDIRS=                        \
      user                      \
      driver                          \
      platform                  \
      libc                      \
      lua                       \
      lwip                      \
      coap                      \
      mqtt                      \
      task                      \
      u8glib                          \
      ucglib                          \
      smart                           \
      modules                   \
      spiffs                          \
      cjson                           \
      crypto                          \
      dhtlib                          \
      tsl2561                         \
      net                       \
      http

endif # } PDIR
```

```
APPDIR = .
LDDIR = ../ld

CCFLAGS += -Os

TARGET_LDFLAGS =                        \
      -nostdlib                   \
      -Wl,-EL                     \
      --longcalls                       \
      --text-section-literals

ifeq ($(FLAVOR),debug)
    TARGET_LDFLAGS += -g -Os
endif

ifeq ($(FLAVOR),release)
    TARGET_LDFLAGS += -Os
endif

LD_FILE = $(LDDIR)/nodemcu.ld

COMPONENTS_eagle.app.v6 =               \
      user/libuser.a                    \
      driver/libdriver.a                \
      platform/libplatform.a                  \
      task/libtask.a                    \
      libc/liblibc.a                    \
      lua/liblua.a                      \
      lwip/liblwip.a                    \
      coap/coap.a                       \
      mqtt/mqtt.a                       \
      u8glib/u8glib.a             \
      ucglib/ucglib.a             \
      smart/smart.a                     \
      spiffs/spiffs.a             \
      cjson/libcjson.a                  \
      crypto/libcrypto.a                \
      dhtlib/libdhtlib.a                \
      tsl2561/tsl2561lib.a              \
      http/libhttp.a                    \
      net/libnodemcu_net.a              \
      modules/libmodules.a              \

# Inspect the modules library and work out which modules need to be
linked.
# For each enabled module, a symbol name of the form
XYZ_module_selected is
# returned. At link time those names are declared undefined, so those
(and
# only those) modules are pulled in.
SELECTED_MODULE_SYMS=$(filter %_module_selected
%module_selected1,$(shell $(NM)
modules/.output/$(TARGET)/$(FLAVOR)/lib/libmodules.a))

LINKFLAGS_eagle.app.v6 =                \
      -Wl,--gc-sections                 \
      -Wl,-Map=mapfile                  \
```

```
      -nostdlib                   \
      -T$(LD_FILE)                      \
      -Wl,@../ld/defsym.rom             \
      -Wl,--no-check-sections           \
      -Wl,--wrap=_xtos_set_exception_handler     \
      -Wl,-static                       \
      $(addprefix -u , $(SELECTED_MODULE_SYMS)) \
      -Wl,--start-group                 \
      -lc                         \
      -lgcc                             \
      -lhal                             \
      -lphy                             \
      -lpp                        \
      -lnet80211                        \
      -lwpa                             \
      -lwpa2                            \
      -lmain                            \
      -lsmartconfig                     \
      -lssl                             \
      -lcrypto                    \
      -lwps                       \
      $(DEP_LIBS_eagle.app.v6)          \
      -Wl,--end-group             \
      -lm

DEPENDS_eagle.app.v6 =                      \
                $(LD_FILE)                  \
           Makefile

##############################################################
# Configuration i.e. compile options etc.
# Target specific stuff (defines etc.) goes in here!
# Generally values applying to a tree are captured in the
#   makefile at its root level - these are then overridden
#   for a subtree within the makefile rooted therein
#

#UNIVERSAL_TARGET_DEFINES =                \

# Other potential configuration flags include:
#     -DTXRX_TXBUF_DEBUG
#     -DTXRX_RXBUF_DEBUG
#     -DWLAN_CONFIG_CCX
CONFIGURATION_DEFINES =    -D__ets__        \
                -DICACHE_FLASH        \
                -DLUA_OPTIMIZE_MEMORY=2     \
                -DMIN_OPT_LEVEL=2     \
                -DLWIP_OPEN_SRC \
                -DPBUF_RSV_FOR_WLAN   \
                -DEBUF_LWIP           \

DEFINES +=                                  \
      $(UNIVERSAL_TARGET_DEFINES)           \
      $(CONFIGURATION_DEFINES)

DDEFINES +=                                 \
```

```
      $(UNIVERSAL_TARGET_DEFINES)           \
      $(CONFIGURATION_DEFINES)


##############################################################
# Recursion Magic - Don't touch this
#
# Each subtree potentially has an include directory
#   corresponding to the common APIs applicable to modules
#   rooted at that subtree. Accordingly, the INCLUDE PATH
#   of a module can only contain the include directories up
#   its parent path, and not its siblings
#
# Required for each makefile to inherit from the parent
#

INCLUDES := $(INCLUDES) -I $(PDIR)include
INCLUDES += -I ./
PDIR := ../$(PDIR)
sinclude $(PDIR)Makefile


.PHONY: FORCE
FORCE:
```

app/include/user_modules.h:

```
#ifndef __USER_MODULES_H__
#define __USER_MODULES_H__

#define LUA_USE_BUILTIN_STRING        // for string.xxx()
#define LUA_USE_BUILTIN_TABLE         // for table.xxx()
#define LUA_USE_BUILTIN_COROUTINE     // for coroutine.xxx()
#define LUA_USE_BUILTIN_MATH          // for math.xxx(), partially work
// #define LUA_USE_BUILTIN_IO              // for io.xxx(), partially
work

// #define LUA_USE_BUILTIN_OS              // for os.xxx(), not work
// #define LUA_USE_BUILTIN_DEBUG
#define LUA_USE_BUILTIN_DEBUG_MINIMAL // for debug.getregistry() and
debug.traceback()

#ifndef LUA_CROSS_COMPILER

// The default configuration is designed to run on all ESP modules
including the 512 KB modules like ESP-01 and only
// includes general purpose interface modules which require at most
two GPIO pins.
// See https://github.com/nodemcu/nodemcu-firmware/pull/1127 for
discussions.
// New modules should be disabled by default and added in alphabetical
```

```
order.
#define LUA_USE_MODULES_ADC
//#define LUA_USE_MODULES_AM2320
//#define LUA_USE_MODULES_APA102
#define LUA_USE_MODULES_BIT
//#define LUA_USE_MODULES_BMP085
//#define LUA_USE_MODULES_BME280
//#define LUA_USE_MODULES_CJSON
//#define LUA_USE_MODULES_COAP
//#define LUA_USE_MODULES_CRYPTO
#define LUA_USE_MODULES_DHT
//#define LUA_USE_MODULES_ENCODER
#define LUA_USE_MODULES_ENDUSER_SETUP // USE_DNS in dhcpserver.h needs
to be enabled for this module to work.
#define LUA_USE_MODULES_FILE
#define LUA_USE_MODULES_GPIO
//#define LUA_USE_MODULES_HTTP
//#define LUA_USE_MODULES_HX711
#define LUA_USE_MODULES_I2C
//#define LUA_USE_MODULES_MDNS
#define LUA_USE_MODULES_MQTT
#define LUA_USE_MODULES_NET
#define LUA_USE_MODULES_NODE
#define LUA_USE_MODULES_OW
//#define LUA_USE_MODULES_PERF
//#define LUA_USE_MODULES_PWM
//#define LUA_USE_MODULES_RC
//#define LUA_USE_MODULES_ROTARY
//#define LUA_USE_MODULES_RTCFIFO
//#define LUA_USE_MODULES_RTCMEM
//#define LUA_USE_MODULES_RTCTIME
//#define LUA_USE_MODULES_SIGMA_DELTA
//#define LUA_USE_MODULES_SNTP
#define LUA_USE_MODULES_SPI
//#define LUA_USE_MODULES_STRUCT
#define LUA_USE_MODULES_TMR
//#define LUA_USE_MODULES_TSL2561
//#define LUA_USE_MODULES_U8G
#define LUA_USE_MODULES_UART
//#define LUA_USE_MODULES_UCG
#define LUA_USE_MODULES_WIFI
//#define LUA_USE_MODULES_WS2801
//#define LUA_USE_MODULES_WS2812

#define LUA_USE_MODULES_WPS // Modulo WPS añadido


#endif  /* LUA_CROSS_COMPILER */
#endif                                                                  /*
__USER_MODULES_H__ */
```

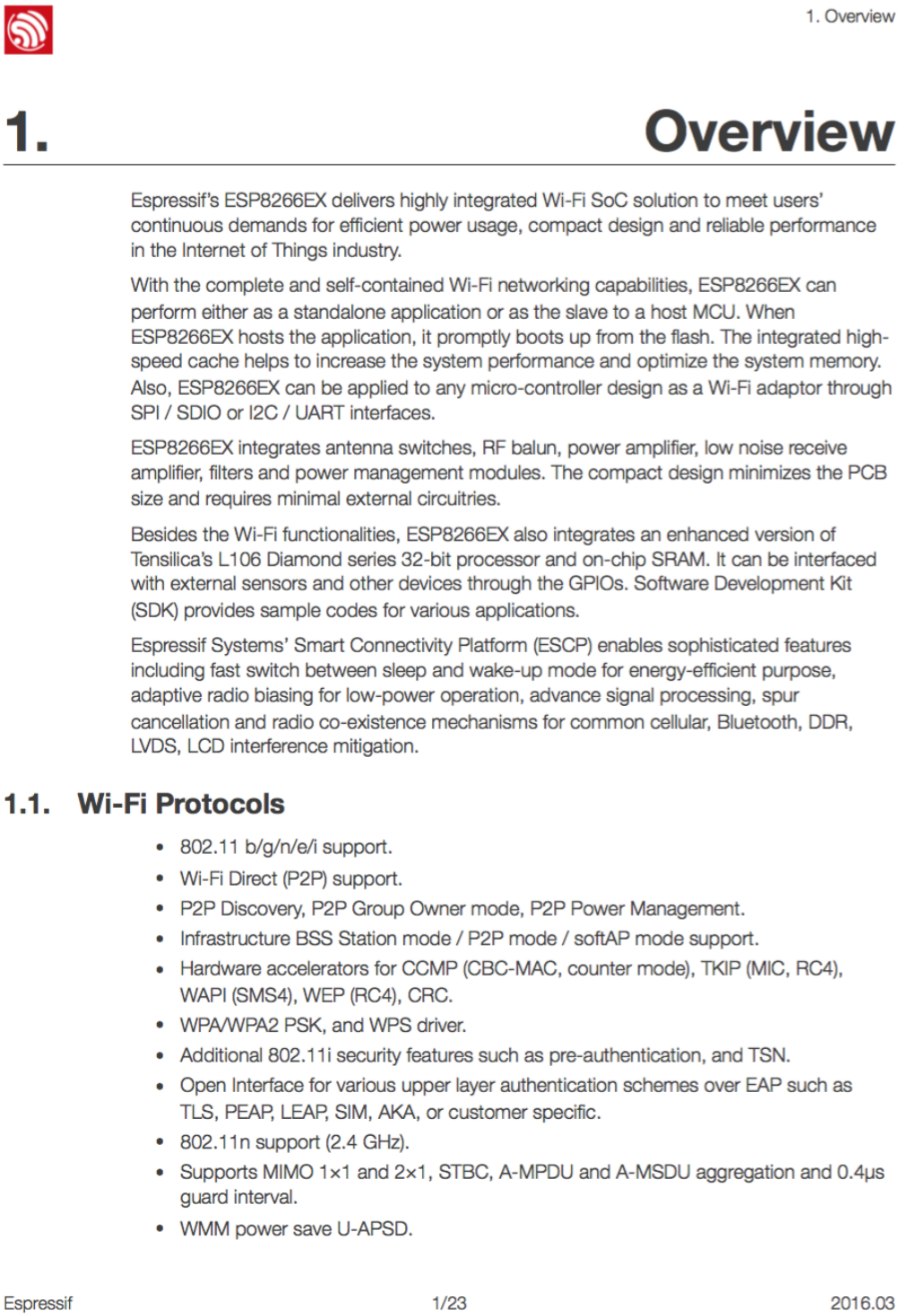
nodemcu-firmware/app/modules/wps.c:

```
// Modulo para el uso del protocolo WPS

#include "module.h"
#include "lauxlib.h"
#include "platform.h"

#include "c_string.h"
#include "c_stdlib.h"
#include "ctype.h"

#include "c_types.h"
#include "user_interface.h"

// Funcion de saludo de wps
static int ICACHE_FLASH_ATTR wps_saludo(lua_State* L)
{
  lua_pushstring(L, "Hola desde el módulo WPS.");
  return 1;
}

// Funcion para deshabilitar el wps
static int ICACHE_FLASH_ATTR wps_disable(lua_State* L)
{
  unsigned int estado;

  estado=wifi_wps_disable();
  return estado;
}

// Funcion para habilitar el wps
static int ICACHE_FLASH_ATTR wps_enable(lua_State* L)
{
  unsigned int estado;

  estado=wifi_wps_enable(WPS_TYPE_PBC);
  return estado;
}

// Funcion de callback del wps
LOCAL void ICACHE_FLASH_ATTR user_wps_status_cb(int status)
{
  switch (status) {
    case WPS_CB_ST_SUCCESS:
      wifi_wps_disable();
      wifi_station_connect();
      break;
    case WPS_CB_ST_FAILED:
    case WPS_CB_ST_TIMEOUT:
      wifi_wps_start();
      break;
  }
}
```

```
// Funcion de llamada
static int ICACHE_FLASH_ATTR wps_set(lua_State* L)
{
  unsigned int estado;

  estado=wifi_set_wps_cb(user_wps_status_cb);
  return estado;
}

// Función para arrancar el wps
static int ICACHE_FLASH_ATTR wps_start(lua_State* L)
{
  unsigned int estado;

  estado=wifi_wps_start();
  return estado;
}

// Module function map
const LUA_REG_TYPE wps_map[] =  {
  { LSTRKEY( "saludo" ),        LFUNCVAL( wps_saludo ) },
  { LSTRKEY( "disable" ),        LFUNCVAL( wps_disable ) },
  { LSTRKEY( "enable" ),        LFUNCVAL( wps_enable ) },
  { LSTRKEY( "set" ),        LFUNCVAL( wps_set ) },
  { LSTRKEY( "start" ),        LFUNCVAL( wps_start ) },
  { LNILKEY, LNILVAL }
};

int luaopen_wps( lua_State *L )
{
 return 0;
}

NODEMCU_MODULE(WPS, "wps", wps_map, luaopen_wps);
```

ANEXO B: DATASHEET OVERVIEW ESP8266

Para más información consultar el datasheet completo en las referencias:



1. Overview

1. Overview

Espressif's ESP8266EX delivers highly integrated Wi-Fi SoC solution to meet users' continuous demands for efficient power usage, compact design and reliable performance in the Internet of Things industry.

With the complete and self-contained Wi-Fi networking capabilities, ESP8266EX can perform either as a standalone application or as the slave to a host MCU. When ESP8266EX hosts the application, it promptly boots up from the flash. The integrated high-speed cache helps to increase the system performance and optimize the system memory. Also, ESP8266EX can be applied to any micro-controller design as a Wi-Fi adaptor through SPI / SDIO or I2C / UART interfaces.

ESP8266EX integrates antenna switches, RF balun, power amplifier, low noise receive amplifier, filters and power management modules. The compact design minimizes the PCB size and requires minimal external circuitries.

Besides the Wi-Fi functionalities, ESP8266EX also integrates an enhanced version of Tensilica's L106 Diamond series 32-bit processor and on-chip SRAM. It can be interfaced with external sensors and other devices through the GPIOs. Software Development Kit (SDK) provides sample codes for various applications.

Espressif Systems' Smart Connectivity Platform (ESCP) enables sophisticated features including fast switch between sleep and wake-up mode for energy-efficient purpose, adaptive radio biasing for low-power operation, advance signal processing, spur cancellation and radio co-existence mechanisms for common cellular, Bluetooth, DDR, LVDS, LCD interference mitigation.

1.1. Wi-Fi Protocols

- 802.11 b/g/n/e/i support.
- Wi-Fi Direct (P2P) support.
- P2P Discovery, P2P Group Owner mode, P2P Power Management.
- Infrastructure BSS Station mode / P2P mode / softAP mode support.
- Hardware accelerators for CCMP (CBC-MAC, counter mode), TKIP (MIC, RC4), WAPI (SMS4), WEP (RC4), CRC.
- WPA/WPA2 PSK, and WPS driver.
- Additional 802.11i security features such as pre-authentication, and TSN.
- Open Interface for various upper layer authentication schemes over EAP such as TLS, PEAP, LEAP, SIM, AKA, or customer specific.
- 802.11n support (2.4 GHz).
- Supports MIMO 1×1 and 2×1, STBC, A-MPDU and A-MSDU aggregation and 0.4μs guard interval.
- WMM power save U-APSD.

Espressif 1/23 2016.03

Figura 30 - Datasheet Overview 1/4

- Multiple queue management to fully utilize traffic prioritization defined by 802.11e standard.
- UMA compliant and certified.
- 802.1h/RFC1042 frame encapsulation.
- Scattered DMA for optimal CPU off load on Zero Copy data transfer operations.
- Antenna diversity and selection (software managed hardware).
- Clock/power gating combined with 802.11-compliant power management dynamically adapted to current connection condition providing minimal power consumption.
- Adaptive rate fallback algorithm sets the optimum transmission rate and Tx power based on actual SNR and packet loss information.
- Automatic retransmission and response on MAC to avoid packet discarding on slow host environment.
- Seamless roaming support.
- Configurable packet traffic arbitration (PTA) with dedicated slave processor based design provides flexible and exact timing Bluetooth co-existence support for a wide range of Bluetooth Chip vendors.
- Dual and single antenna Bluetooth co-existence support with optional simultaneous receive (Wi-Fi/Bluetooth) capability.

Figura 31 - Datasheet Overview 2/4

1.2. Main Technical Specifications

Table 1-1. Main Technical Specifications

Categories	Items	Parameters
Wi-Fi	Standards	FCC/CE/TELEC/SRRC
	Protocols	802.11 b/g/n/e/i
	Frequency Range	2.4 G ~ 2.5 G (2400M ~ 2483.5M)
	Tx Power	802.11 b: +20 dBm
		802.11 g: +17 dBm
		802.11 n: +14 dBm
	Rx Sensitivity	802.11 b: -91 dbm (11 Mbps)
		802.11 g: -75 dbm (54 Mbps)
		802.11 n: -72 dbm (MCS7)
	Antenna	PCB Trace, External, IPEX Connector, Ceramic Chip
Hardware	CPU	Tensilica L106 32-bit micro controller
	Peripheral Interface	UART/SDIO/SPI/I2C/I2S/IR Remote Control
		GPIO/ADC/PWM
	Operating Voltage	3.0 V ~ 3.6 V
	Operating Current	Average value: 80 mA
	Operating Temperature Range	-40°C ~ 125°C
	Storage Temperature Range	-40°C ~ 125°C
	Package Size	QFN32-pin (5 mm x 5 mm)
	External Interface	-
Software	Wi-Fi Mode	station/softAP/SoftAP+station
	Security	WPA/WPA2
	Encryption	WEP/TKIP/AES
	Firmware Upgrade	UART Download / OTA (via network)
	Software Development	Supports Cloud Server Development / Firmware and SDK for fast on-chip programming
	Network Protocols	IPv4, TCP/UDP/HTTP/FTP
	User Configuration	AT Instruction Set, Cloud Server, Android/iOS App

1.3. Applications

- Home Appliances
- Home Automation
- Smart Plugs and Lights
- Mesh Network

Figura 32 - Datasheet Overview 3/4

Figura 33 - Datasheet Overview 4/4